НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
_інформаційних систем і технологій__
(назва кафедри)

_____ /_____Швиденко М.З._____/
(підпис) (ПІБ)

“___” _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Інформаційна система для автоматизації бізнес-процесів малого підприємства»

Спеціальність 122 – «Комп’ютерні науки»

Гарант освітньої програми
_____д.е.н., професор_____ _____ _____Руденський Р.А.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
_____ _____ _Золотуха Роман Андрійович_
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _____Івашко Максим Віталійович___
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
__інформаційних систем і технологій__
(назва кафедри)

___к.е.н., доцент_____Швиденко М.З.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"___" _____ 2025 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Івашко Максим Віталійович

Спеціальність 122 – «Комп'ютерні науки»

1. Тема бакалаврської кваліфікаційної роботи «Інформаційна система для автоматизації бізнес-процесів малого підприємства» затверджена наказом ректора НУБіП України від 16.12.2024 № 2246"С"
2. Термін подання завершеної роботи на кафедру_____
(рік, місяць, число)
3. Вихідні дані до роботи: отримання відомостей, цифрових показників щодо закупівлі товарів у постачальників та продажу товарів клієнтам.
4. Перелік питань, що розглядаються:
   - Аналіз проблемної області
   - Моделювання предметної області
   - Проєктування програмної системи
   - Впровадження та експлуатація системи

Керівник кваліфікаційної роботи

д. філос. н. (спец. 122), ст. викл.		Золотуха Р.А.
(науковий ступінь та вчене звання)	(підпис)	(ПІБ)

Завдання прийняв до виконання		Івашко М.В.
	(підпис)	(ПІБ)

Дата отримання завдання "___"_____ 2025 р.

ЗМІСТ

ВСТУП

У сучасному бізнес-середовищі, яке швидко розвивається, невеликим підприємствам часто важко ефективно керувати процесами закупівель і продажів. Багато хто досі покладаються на застарілі ручні методи, такі як електронні таблиці та паперові записи, які схильні до людських помилок, неефективності та втрати даних. Ці традиційні підходи ускладнюють відстеження транзакцій, управління відносинами з постачальниками та ведення точного обліку запасів. У міру зростання бізнесу відсутність автоматизації призводить до затримок, операційних вузьких місць і збільшення витрат, що ускладнює збереження конкурентоспроможності.

Автоматизація бізнес-процесів є важливим кроком для підвищення ефективності роботи малих підприємств.

Метою даного дослідження є оптимізація управління постачальниками, клієнтами, товарами, а також процеси закупівлі та продажу. Для досягнення цієї мети необхідно було здійснити всебічний аналіз існуючих методів цифровізації діяльності малих підприємств, окреслити особливості організації їх внутрішніх операцій, пов’язаних із товарообігом та клієнтською базою, а також дослідити типові труднощі, які виникають при використанні традиційних облікових засобів.

Актуальність цієї роботи зумовлена зростаючою потребою малого бізнесу в цифрових рішеннях, що дозволяють конкурувати в умовах сучасного ринку. Багато підприємств досі використовують ручні методи обліку, що не лише уповільнює робочі процеси, а й створює ризики втрати даних або неточностей у звітах. Впровадження автоматизованої системи сприятиме підвищенню продуктивності, покращенню взаємодії з постачальниками та клієнтами, а також сприятиме ефективному управлінню ресурсами підприємства.

Об'єктом дослідження є процеси управління закупівлями, продажами, обліком товарів і взаємодією з клієнтами та постачальниками у малих підприємствах.

Предметом дослідження є методи та засоби автоматизації бізнес-процесів малого підприємства за допомогою інформаційної системи, що базується на веб-технологіях.

Щоб вирішити ці проблеми, це дослідження досліджує розробку інформаційної системи для автоматизації бізнес-процесів малого підприємства, призначеної для оптимізації закупівель, управління замовленнями та взаємодії з клієнтами. Система забезпечує централізовану платформу для керування постачальниками, продуктами, клієнтами та фінансовими операціями. Замінивши ручні процеси цифровим рішенням, підприємства можуть підвищити операційну точність, покращити процес прийняття рішень і зменшити адміністративний тягар.

Це дослідження зосереджено на розробці функціонального та зручного рішення, яке спрощує важливі бізнес-операції. Система міститиме інструменти керування постачальниками та клієнтами, каталогізацію продукції, автоматизовану обробку замовлень і функції звітності в реальному часі. Створений на основі PHP із структурою Symfony та MySQL як базою даних, він забезпечує масштабованість, безпеку та легкість обслуговування.

Основна частина роботи структурована у три ключові розділи. У першому розділі виконано аналіз проблемної області, виявлено недоліки існуючих рішень і обґрунтовано доцільність розробки власної системи. Другий розділ присвячено моделюванню предметної області — у ньому сформовано вимоги до системи, розглянуто основні сценарії її використання та побудовано відповідні UML-діаграми. Третій розділ зосереджено на проєктуванні програмної системи: описано архітектуру, структуру бази даних, ключові компоненти та особливості реалізації функціоналу із використанням сучасних веб-технологій.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис предметної області

Малі підприємства відіграють вирішальну роль в економічному розвитку, але вони часто стикаються з неефективними бізнес-процесами через обмежені ресурси та застарілі методи управління. Однією з найбільш критичних проблем, з якими вони стикаються, є відсутність автоматизованої системи для управління закупівлями, запасами та продажами. Багато компаній досі покладаються на ручне введення даних, електронні таблиці чи паперову документацію, що призводить до помилок, затримок і невідповідностей. Ця неефективність ускладнює відстеження транзакцій постачальників, моніторинг рівня запасів у режимі реального часу та ефективну обробку замовлень клієнтів.

Відсутність централізованої інформаційної системи призводить до фрагментації бізнес-операцій, коли різні аспекти закупівель і продажів управляються незалежно, часто без синхронізації. Це створює вузькі місця в обробці замовлень, збільшує ризик дефіциту або надлишку запасів і ускладнює фінансове планування. Крім того, без належної автоматизації прийняття рішень стає реактивним, а не проактивним, оскільки власникам бізнесу бракує точних, актуальних даних для прогнозування та розробки стратегії.

Іншою важливою проблемою є проблема підтримки ефективних відносин із постачальниками та клієнтами. У ручній системі відстеження взаємодій, історії платежів і деталей контракту потребує додаткових зусиль, що може призвести до непорозуміння або втрачених можливостей для переговорів. Подібним чином, без упорядкованої обробки замовлень на продаж підприємства можуть мати проблеми із затримками поставок, неточними рахунками-фактурами та незадоволеністю клієнтів [3].

Зростаючий попит на автоматизацію бізнесу підкреслює потребу в надійному та ефективному цифровому рішенні, призначеному для малих підприємств. Завдяки інтеграції інформаційної системи, яка автоматизує

процеси закупівель і продажів, підприємства можуть підвищити операційну ефективність, зменшити адміністративне навантаження та підвищити загальну прибутковість. Це дослідження зосереджено на виявленні ключових проблемних точок у діяльності малого бізнесу та пропозиції системи, яка вирішує ці проблеми за допомогою технологічних рішень.

Автоматизація бізнес-процесів стала вирішальним фактором підвищення ефективності та конкурентоспроможності малих підприємств. На відміну від великих корпорацій, які мають доступ до великих ресурсів і систем управління на рівні підприємства, малі підприємства часто покладаються на ручні робочі процеси, електронні таблиці або базове бухгалтерське програмне забезпечення для керування закупівлями, продажами та запасами. Однак такі підходи схильні до помилок, неефективності та затримок у прийнятті рішень. Інформаційна система для автоматизації бізнес-процесів малого підприємства спрямована на вирішення цих проблем шляхом інтеграції ключових операційних функцій в єдину цифрову платформу, оптимізації управління робочими процесами та використання ресурсів.

Предмет цього дослідження охоплює основні процеси закупівель і продажів у малому бізнесі. Закупівлі включають керування постачальниками, відстеження замовлень на купівлю, моніторинг поставок і обробку платежів. Процеси продажів, у свою чергу, включають керування замовленнями клієнтів, створення рахунків-фактур, відстеження платежів та оновлення рівня запасів. Ефективна автоматизація в цих областях забезпечує безперебійну координацію між постачальниками, клієнтами та внутрішніми бізнес-операціями, зменшуючи паперову роботу, покращуючи точність і підвищуючи продуктивність [3].

Важливим аспектом автоматизації бізнес-процесів є централізація даних, де вся інформація про постачальників, клієнтів, продукти та транзакції зберігається в структурованій базі даних. Це дає змогу в режимі реального часу відстежувати рівень запасів, автоматичні повідомлення про поповнення запасів та ефективне фінансове управління. Система також сприяє підтримці прийняття рішень, надаючи аналітичну інформацію про тенденції закупівель, попит

клієнтів і ефективність постачальників, допомагаючи підприємствам робити обґрунтований стратегічний вибір.

Основні програмні функції системи:

- автентифікація користувачів і контроль доступу;
- введення даних про закупівлі;
- шифрування даних і безпечне зберігання;
- керування робочим процесом закупівель;
- аналіз закупівель і звітність;
- моніторинг безпеки та журналювання;
- керування виправленнями безпеки;
- аудит безпеки та відповідність.

Включаючи ці програмні функції, програмна система підвищує безпеку закупівельної діяльності, захищаючи дані, контролюючи доступ, автоматизуючи процеси, надаючи аналітичну інформацію та забезпечуючи відповідність найкращим практикам і нормам безпеки.

Впроваджуючи веб-рішення, побудоване за допомогою PHP із структурою Symfony і MySQL, малі підприємства можуть отримати вигоду від доступної та масштабованої платформи, яка підтримує багатокористувацький доступ, дозволи на основі ролей і безпечне керування даними. Запровадження такої системи не тільки зменшує операційні витрати, але й підвищує здатність малих підприємств ефективно реагувати на зміни ринку. Це дослідження досліджує, як цифрова трансформація в управлінні закупівлями та продажами може створити більш структуроване, безпомилкове та орієнтоване на зростання бізнес-середовище.

1.2 Аналіз наявних інформаційних технологій

Сучасний ринок пропонує різноманітні програмні рішення для управління закупівлями, кожне з яких має свої особливості, функціональні можливості та рівень безпеки. Окрім програмної системи, що розробляється для

відображення та аналізу купівельної діяльності підприємства, існують альтернативні програми, які також спрямовані на оптимізацію закупівельних процесів, управління постачальниками та аналіз витрат.

Одним із ключових рішень є програмне забезпечення для керування закупівлями, яке дозволяє підприємствам автоматизувати основні процеси – створення заявок, роботу з постачальниками, оформлення замовлень та відстеження рахунків-фактур. Такі системи забезпечують можливість формування звітності, контролю витрат та ведення історії взаємодії з постачальниками. Проте, хоча багато з них включають базові механізми безпеки, такі як автентифікація користувачів і контроль доступу, питання захисту даних може не бути їхнім основним фокусом [4].

Більш комплексний підхід до управління бізнес-процесами пропонують системи планування ресурсів підприємства (ERP). Вони інтегрують закупівлі, управління запасами, фінанси та інші аспекти діяльності в єдину платформу. Такі системи містять модулі для керування закупівлями, що дозволяє підприємствам контролювати всі етапи цього процесу, отримувати детальні аналітичні звіти та оптимізувати взаємодію між відділами. ERP-системи зазвичай мають розширені механізми безпеки, включаючи контроль доступу на основі ролей та складніші методи автентифікації, але їхня загальна спрямованість виходить далеко за межі лише закупівельної діяльності.

Ще однією категорією рішень є інструменти бізнес-аналітики (BI), які спеціалізуються на зборі, аналізі та візуалізації даних, зокрема закупівельної інформації. Вони надають інформаційні панелі, інтерактивні звіти та можливості глибокого аналізу для виявлення тенденцій у купівельній діяльності та оцінки ефективності співпраці з постачальниками. Хоча такі інструменти не завжди охоплюють весь процес закупівель, вони можуть значно підсилити аналітичний аспект управління закупівлями. Безпека BI-систем залежить від конкретного інструменту та його інтеграції з іншими програмними продуктами, що може впливати на рівень захисту даних.

Таким чином, на ринку представлено широкий спектр рішень для автоматизації закупівельних процесів, які допомагають підприємствам оптимізувати свою діяльність. Вибір відповідного програмного забезпечення залежить від потреб компанії, рівня інтеграції із суміжними бізнес-процесами та вимог до безпеки даних.

Розглянемо існуючі рішення предметної області.

SAP Ariba — це провідне хмарне рішення для управління закупівлями та ланцюжками поставок, яке допомагає компаніям оптимізувати процеси закупівель, керувати відносинами з постачальниками та оптимізувати управління витратами. Будучи частиною екосистеми SAP, Ariba надає комплексну платформу для автоматизації робочих процесів закупівель, покращення співпраці з постачальниками та покращення видимості операцій із закупівель [1].

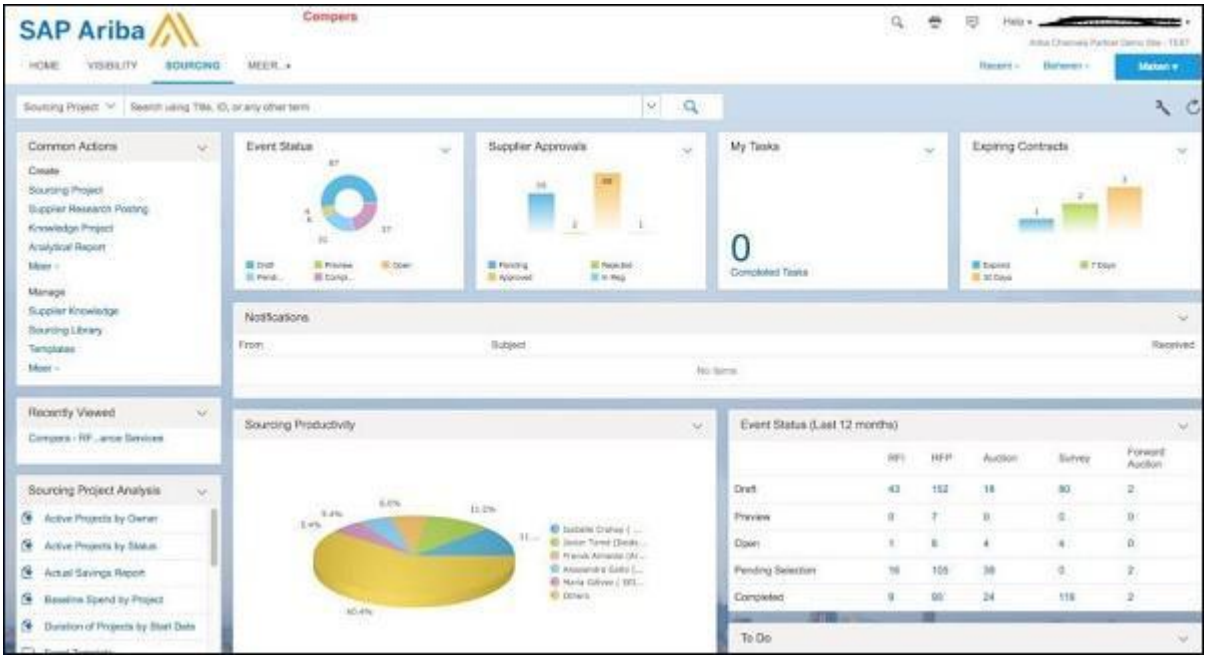

Рис. 1.1 Сервіс SAP Ariba

Однією з ключових переваг SAP Ariba є її здатність об’єднувати підприємства з великою глобальною мережею постачальників через Ariba Network, один із найбільших у світі цифрових ринків для B2B комерції. Ця мережа дозволяє компаніям знаходити нових постачальників, укладати контракти та ефективно автоматизувати операції. Завдяки інтегрованим

інструментам для процесів «джерело-оплата» та «закупівля-оплата» організації можуть керувати всім: від вибору постачальника та керування контрактами до виставлення рахунків і платежів на одній платформі.

SAP Ariba пропонує розширені функції аналізу витрат і управління ризиками, що дозволяє підприємствам отримати детальну інформацію про свої закупівельні дані, визначити можливості економії коштів і зменшити ризики в ланцюжку поставок. Крім того, платформа надає інструменти відповідності та управління для забезпечення дотримання корпоративної політики та галузевих норм [1].

Система базується на хмарі, що забезпечує гнучкість, масштабованість і повну інтеграцію з існуючими рішеннями ERP, включаючи SAP S/4HANA. Це дозволяє підприємствам будь-якого розміру підвищити ефективність закупівель, зменшити операційні витрати та покращити співпрацю з постачальниками. Використовуючи штучний інтелект, автоматизацію та аналітику в реальному часі, SAP Ariba дає організаціям можливість приймати більш розумні рішення щодо закупівель і досягати більшої гнучкості бізнесу.

Розглянемо інше програмне рішення на ринку.

Precoro — це хмарне програмне забезпечення для керування закупівлями, призначене для оптимізації та автоматизації процесів закупівель для підприємств будь-якого розміру. Він зосереджений на підвищенні ефективності закупівель, покращенні фінансового контролю та полегшенні співпраці між командами та постачальниками. Завдяки інтуїтивно зрозумілому інтерфейсу та надійній функціональності Precoro спрощує робочий процес із закупівель від заявки до оплати [2].

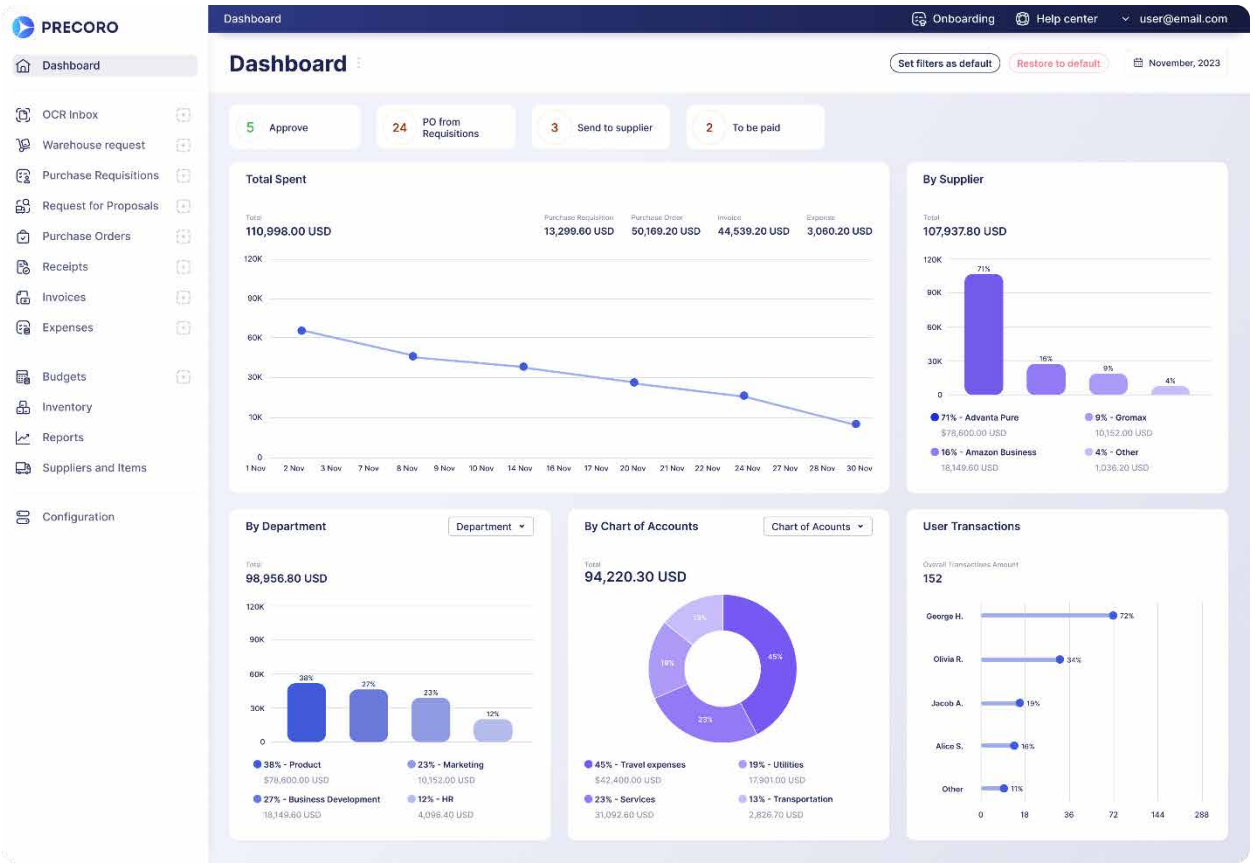

Рис. 1.2 Інформаційна система «Precoro»

Однією з ключових особливостей Precoro є керування замовленнями на купівлю, яке дозволяє користувачам легко створювати, відстежувати та затверджувати замовлення на купівлю. Система дозволяє підприємствам визначати робочі процеси затвердження, гарантуючи, що всі покупки перевіряються та авторизуються відповідно до політики компанії. Ця функція допомагає контролювати бюджет і запобігає несанкціонованим витратам.

Precoro також пропонує інструменти керування постачальниками, які дозволяють організаціям ефективно залучати та оцінювати постачальників. Користувачі можуть зберігати інформацію про постачальників, відстежувати показники ефективності та керувати контрактами на одній централізованій платформі. Ця функція сприяє прозорості та допомагає підприємствам будувати міцні стосунки зі своїми постачальниками.

Крім того, Precoro надає функції для обробки рахунків-фактур і керування витратами, що дозволяє організаціям автоматизувати затвердження рахунків-фактур і відстежувати витрати в режимі реального часу. Програмне

забезпечення інтегрується з різними системами бухгалтерського обліку, забезпечуючи безперебійний потік даних і зменшуючи помилки ручного введення даних [2].

Можливості аналітики та звітності платформи дозволяють компаніям отримати інформацію про свою закупівельну діяльність, визначити можливості економії та оптимізувати витрати. За допомогою настроюваних інформаційних панелей і звітів користувачі можуть відстежувати ключові показники ефективності (KPI) і приймати рішення на основі даних.

Загалом Precoro розроблено для підвищення ефективності закупівель, покращення відповідності та забезпечення кращої видимості витрат, що робить його цінним інструментом для організацій, які прагнуть оптимізувати свої процеси закупівель. Його зручний інтерфейс у поєднанні з потужними функціями дозволяє підприємствам швидко адаптуватися до мінливих потреб закупівель і досягти більшої ефективності роботи.

1.3 Постановка завдання

Основною метою цього проєкту є розробка інформаційної системи для автоматизації бізнес-процесів на малих підприємствах, приділяючи особливу увагу оптимізації управління закупівлями та продажами. Ця система надасть власникам бізнесу та співробітникам ефективні інструменти для управління відносинами з постачальниками, відстеження запасів, обробки замовлень на купівлю та продаж, а також аналізу фінансових даних. Завдяки автоматизації цих процесів система прагне зменшити ризики, пов’язані з ручним введенням даних, помилками в обробці замовлень і неефективним управлінням запасами, що зрештою призводить до економії коштів і підвищення ефективності роботи.

Для досягнення цієї мети важливо створити надійну базу даних, яка зберігає важливу інформацію, включаючи відомості про постачальників, каталоги продуктів, замовлення на купівлю, замовлення на продаж і фінансові

операції. Розроблене програмне забезпечення дозволить малим підприємствам отримати швидкий і легкий доступ до важливої інформації, сприяючи:

- ефективне управління постачальниками та клієнтами, включаючи адаптацію, відстеження продуктивності та зв’язок;
- спрощена обробка замовлень, що дозволяє створювати, відстежувати та затверджувати замовлення на купівлю та продаж;
- моніторинг рівня запасів у режимі реального часу, що допомагає запобігти вичерпанню та надлишку запасів;
- комплексні можливості звітності та аналітики, які дають змогу зрозуміти моделі витрат, ефективність постачальників і загальну ефективність закупівель.

Система матиме інтуїтивно зрозумілий інтерфейс користувача, який дозволить користувачам легко переміщатися між різними функціями, пов’язаними з управлінням закупівлями та продажами. Крім того, він включатиме інструменти для створення звітів про стан замовлень, продуктивність постачальника та показники запасів, що дозволить власникам бізнесу приймати обґрунтовані рішення та оптимізувати свої операційні процеси.

Впровадивши цю інформаційну систему, малі підприємства можуть покращити процеси закупівель і продажів, покращити співпрацю між членами команди та досягти більшої ефективності в управлінні своїми бізнес-операціями. Ця цифрова трансформація дозволить їм ефективніше конкурувати на ринку, підвищити прибутковість і підтримувати стійке зростання.

1.4 Функціональні та нефункціональні вимоги

Функціональні вимоги:

1. система повинна дозволяти користувачам створювати та керувати обліковими записами з контролем доступу на основі ролей, гарантуючи, що користувачі мають відповідні дозволи на основі

їхніх ролей (наприклад, адміністратор, офіцер із закупівель, торговий представник);

2. користувачі повинні мати можливість додавати, редагувати та видаляти інформацію про постачальника, включаючи контактні дані, пропоновані послуги та показники ефективності;
3. система повинна забезпечувати централізовану базу даних для зберігання всієї інформації, пов’язаної з постачальником;
4. користувачі повинні мати можливість створювати та керувати профілями клієнтів, включаючи контактну інформацію, історію замовлень та умови оплати;
5. система повинна дозволяти користувачам додавати, оновлювати та видаляти продукти з інвентарю, включаючи такі деталі, як назва продукту, опис, ціна та рівень запасів;
6. користувачі повинні мати можливість створювати, переглядати, редагувати та затверджувати замовлення на купівлю в системі;
7. система повинна забезпечувати відстеження статусу замовлення на купівлю від створення до виконання;
8. система повинна дозволяти користувачам створювати, переглядати, редагувати та обробляти замовлення на продаж, включаючи створення рахунків-фактур і відстеження статусу платежу;
9. користувачі матимуть доступ до рівня запасів у режимі реального часу та сповіщень про низькі запаси товарів або їх немає в наявності;
10. система повинна сприяти автоматичному оновленню рівнів запасів після створення замовлень на купівлю або продаж;
11. система повинна генерувати звіти про діяльність із закупівель і продажів, роботу постачальника та стан запасів;

12. користувачі повинні мати можливість налаштовувати звіти на основі конкретних критеріїв і експортувати їх у різні формати (наприклад, PDF, Excel);
13. система повинна надавати сповіщення про важливі події, такі як очікування схвалення, низький рівень запасів і оновлення статусу замовлення.

Нефункціональні вимоги:

1. система повинна мати інтуїтивно зрозумілий інтерфейс користувача, у якому легко орієнтуватися, дозволяючи користувачам виконувати завдання з мінімальною підготовкою;
2. система повинна підтримувати щонайменше 100 одночасних користувачів без зниження продуктивності;
3. система повинна забезпечувати обробку даних у реальному часі та час відповіді менше 2 секунд на запити користувачів;
4. система має бути розроблена з урахуванням майбутнього зростання, дозволяючи додавати нових користувачів, постачальників і продукти без потреби в суттєвій зміні конфігурації;
5. система має реалізовувати безпечні механізми автентифікації, включаючи шифрування пароля та багатофакторну автентифікацію;
6. дані користувача та інформація про транзакції повинні бути захищені від несанкціонованого доступу за допомогою рольового контролю доступу та безпечних протоколів передачі даних (наприклад, HTTPS) [5];
7. система повинна забезпечувати високу доступність із часом безвідмовної роботи щонайменше 99,5%, зводячи до мінімуму збої в бізнес-операціях;

8. система повинна бути розроблена для простого обслуговування та оновлень, дозволяючи швидко впроваджувати виправлення помилок і покращувати функції;
9. система має бути сумісною з поширеними веб-браузерами (наприклад, Chrome, Firefox, Safari) і мобільними пристроями, забезпечуючи доступність для користувачів на різних платформах;
10. система повинна здійснювати регулярне резервне копіювання даних і забезпечувати механізм відновлення для відновлення даних у разі збою системи або втрати даних.

Визначаючи ці функціональні та нефункціональні вимоги, інформаційна система ефективно задовольнить потреби малих підприємств, надаючи надійне рішення для автоматизації їхніх бізнес-процесів.

1.5 Висновки до 1 розділу

У результаті проведеного аналізу предметної області було встановлено, що малі підприємства часто стикаються з труднощами в управлінні бізнес-процесами через використання застарілих або неефективних методів обліку та обробки інформації. Розгляд основних аспектів функціонування таких підприємств дозволив виявити типові проблеми, пов’язані з обліком товарів, роботою з постачальниками та клієнтами, а також з організацією процесів закупівлі та реалізації продукції.

Аналіз існуючих інформаційних технологій, що застосовуються у цій сфері, показав, що хоча на ринку представлені різноманітні рішення, більшість з них є або надто складними для малого бізнесу, або не враховують його специфіки. Це створює необхідність у розробці більш гнучкої, адаптивної та доступної інформаційної системи, яка б задовольняла потреби користувачів із різним рівнем технічної підготовки.

У рамках постановки завдання були окреслені основні цілі, які має реалізовувати система, а також сформульовані функціональні та

нефункціональні вимоги до неї. Система повинна забезпечувати ефективне керування клієнтською базою, обробку замовлень, облік товарів і фінансових операцій. Водночас вона має бути зручною у використанні, безпечною, масштабованою та відповідати сучасним стандартам веб-розробки.

Таким чином, результати першого розділу стали підґрунтям для побудови моделі предметної області та подальшого проєктування інформаційної системи для автоматизації бізнес-процесів малого підприємства.

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Об’єктне та функціональне моделювання

Уніфікована мова моделювання (UML) — це стандартизована мова моделювання, яка широко використовується в розробці програмного

забезпечення для візуалізації, визначення, конструювання та документування різноманітних артефактів системи. Він надає набір методів графічної нотації, які полегшують створення абстрактних моделей, дозволяючи розробникам, архітекторам і зацікавленим сторонам краще зрозуміти структуру та поведінку програмних систем.

UML містить різноманітні діаграми, які можна розділити на дві основні категорії: структурні діаграми та діаграми поведінки. Структурні діаграми зосереджені на статичних аспектах системи, підкреслюючи організацію її компонентів і зв'язки між ними. Наприклад, діаграми класів ілюструють різні класи в системі, демонструючи їхні атрибути, методи та взаємозв'язки. Ці діаграми є фундаментальними для об'єктно-орієнтованого проєктування, оскільки вони забезпечують план архітектури системи. Іншою важливою структурною схемою є діаграма компонентів, яка зображує різні компоненти або модулі системи разом із їхніми залежностями, допомагаючи з'ясувати, як різні частини взаємодіють одна з одною. Діаграми розгортання додатково ілюструють фізичне розгортання артефактів на вузлах, таких як сервери та пристрої, і дають уявлення про те, як компоненти програмного забезпечення розподіляються між обладнанням. Крім того, пакетні діаграми групують пов'язані класи та компоненти в узгоджені пакети, ефективно керуючи складністю великих систем [6].

З іншого боку, діаграми поведінки відображають динамічні аспекти системи, зосереджуючись на тому, як вона поводиться з часом. Діаграми варіантів використання представляють функціональні вимоги системи, ілюструючи взаємодію між користувачами (акторами) і варіанти використання системи. Цей тип діаграми допомагає зацікавленим сторонам зрозуміти можливості системи та те, як користувачі взаємодіятимуть з нею. Діаграми послідовності пропонують більш детальне уявлення про взаємодію між об'єктами в часі, показуючи обмін повідомленнями в конкретних сценаріях і візуалізуючи потік керування та даних. Діаграми діяльності описують робочі процеси в системі або певні процеси, ілюструючи послідовність дій, точки

прийняття рішень і паралельні процеси. Ці діаграми особливо корисні для моделювання складних бізнес-процесів. Діаграми кінцевого автомата представляють різні стани об’єкта та переходи між цими станами у відповідь на події, ефективно моделюючи поведінку об’єктів, стан яких змінюється. Діаграми співпраці зосереджені на взаємодії між об’єктами з точки зору повідомлень, якими вони обмінюються, наголошуючи на взаємозв’язках, а не на часовій послідовності.

Загалом UML служить потужним інструментом спілкування між зацікавленими сторонами, сприяючи спільному розумінню системи та її вимог. Надаючи візуальні представлення, які спрощують складні системи, UML покращує обговорення дизайну та допомагає в документуванні та майбутньому обслуговуванні. Його універсальність робить його застосовним для різних методологій розробки програмного забезпечення, включно з гнучкою, каскадною розробкою та розробкою на основі моделі. Підводячи підсумок, можна сказати, що UML є важливою структурою для проєктування та документації програмного забезпечення, сприяючи співпраці та гарантуючи, що всі зацікавлені сторони мають чітке та послідовне розуміння системи, що розробляється [6].

Діаграма — це візуальне представлення, яке передає інформацію, концепції або процеси, спрямоване на спрощення складних ідей для легшого розуміння. Діаграми широко використовуються в різних галузях, зокрема в математиці, інженерії, науці, бізнесі та розробці програмного забезпечення, для ілюстрації зв’язків, ієрархій, робочих процесів або структур.

Різні типи діаграм служать певним цілям. Блок-схеми, наприклад, ілюструють потік процесу або системи за допомогою символів для представлення різних кроків, рішень і дій. Вони особливо ефективні для візуалізації робочих процесів і розуміння послідовності операцій. У сфері розробки програмного забезпечення діаграми Уніфікованої мови моделювання (UML) використовуються для моделювання структури та поведінки програмних систем. Загальні діаграми UML, такі як діаграми класів, діаграми послідовності

та діаграми варіантів використання, зосереджуються на різних аспектах проєктування системи.

Організаційні діаграми зображують структуру організації, показуючи зв'язки та ієрархію між різними ролями, відділами чи командами. Інтелектуальні карти, з іншого боку, візуально організовують інформацію навколо центральної ідеї, з гілками, що представляють пов'язані поняття. Цей формат часто використовують для мозкового штурму та впорядкування думок.

У сфері IT і телекомунікацій мережеві діаграми ілюструють структуру мережі, включаючи пристрої, з'єднання та шляхи зв'язку. Діаграми Ганта — ще один тип діаграм, який часто використовується в управлінні проєктами; вони представляють часову шкалу проєкту, показуючи завдання, їх тривалість і залежності [7].

Загалом, діаграми є безцінними інструментами для ефективного спілкування. Вони спрощують складну інформацію, підкреслюють зв'язки та надають чіткі огляди тем. Включаючи візуальні елементи, діаграми покращують розуміння та запам'ятовування, що робить їх дуже ефективними для презентацій, звітів і спільних зусиль.

2.1.1 Діаграма прецедентів. Діаграма варіантів використання — це візуальне представлення, яке ілюструє взаємодію між користувачами (акторами) і системою, підкреслюючи функціональні вимоги системи. Він є частиною Уніфікованої мови моделювання (UML) і служить інструментом для розуміння й аналізу поведінки системи з точки зору користувача. Діаграми варіантів використання особливо корисні на ранніх стадіях розробки програмного забезпечення, оскільки вони допомагають визначити та уточнити ключові функціональні можливості, які має надавати система.

В основі діаграми варіантів використання знаходяться актори та варіанти використання. Актори представляють різних користувачів або зовнішні системи, які взаємодіють із системою, що розробляється. Їх можна класифікувати на основних дійових осіб, які ініціюють взаємодію, і вторинних дійових осіб, які надають підтримку або додаткову функціональність.

Наприклад, у програмі електронної комерції основними учасниками можуть бути клієнти та адміністратори, тоді як платіжний шлюз може бути вторинним учасником [8].

Варіанти використання описують конкретні дії або послуги, які система виконує у відповідь на запит актора. Кожен варіант використання окреслює конкретну мету, яку хоче досягти учасник, наприклад розміщення замовлення, керування запасами або створення звіту. Варіанти використання зазвичай представлені овалами на діаграмі з описовими назвами, які підсумовують функціональність.

Відносини між акторами та варіантами використання зображені за допомогою ліній, які їх з'єднують. Ці відносини можуть вказувати на різні типи взаємодії:

- асоціація: проста лінія, що з'єднує актора з варіантом використання, вказуючи, що актор бере участь у варіанті використання;
- include: зв'язок, який показує, що варіант використання включає функціональні можливості іншого варіанту використання. Це використовується для представлення загальних функціональних можливостей, які можна повторно використовувати в кількох випадках використання;
- extend: зв'язок, який вказує на те, що варіант використання може бути розширений іншим варіантом використання, зазвичай для додавання необов'язкової поведінки або додаткових функцій.

Діаграми варіантів використання забезпечують кілька переваг у процесі розробки програмного забезпечення. Вони полегшують спілкування між зацікавленими сторонами, представляючи високорівневий погляд на функціональність системи в легко зрозумілому форматі. Ця ясність допомагає гарантувати, що всі учасники мають спільне розуміння цілей системи та взаємодії користувачів. Крім того, діаграми варіантів використання служать

основою для більш детальних специфікацій, керуючи розробкою інших діаграм UML і технічної документації.

Для цієї системи була розроблена діаграма прецедентів, яка містить одного актора – "Бухгалтер".

Актор "Бухгалтер" має кілька прецедентів, серед яких:

- закупівля товарів;
- пошук постачальника;
- додавання нових товарів до складу;
- перевірка складу;
- видалення товару;
- редагування товарів на складі;
- формування звіту.

Прецедент "Закупити товари" пов'язаний з іншими прецедентами через зв'язок "include", зокрема з прецедентами "Назва товару" та "Кількість товару". Цей зв'язок вказує на те, що для виконання закупівлі необхідно уточнити, які товари і в якій кількості будуть придбані.

Крім того, прецедент "Додати нові товари до складу" включає анотації, такі як "Дата прибуття", "Назва товару" та "Кількість". Ці деталі допомагають забезпечити повну інформацію про нові надходження на склад, що сприяє ефективному управлінню запасами.

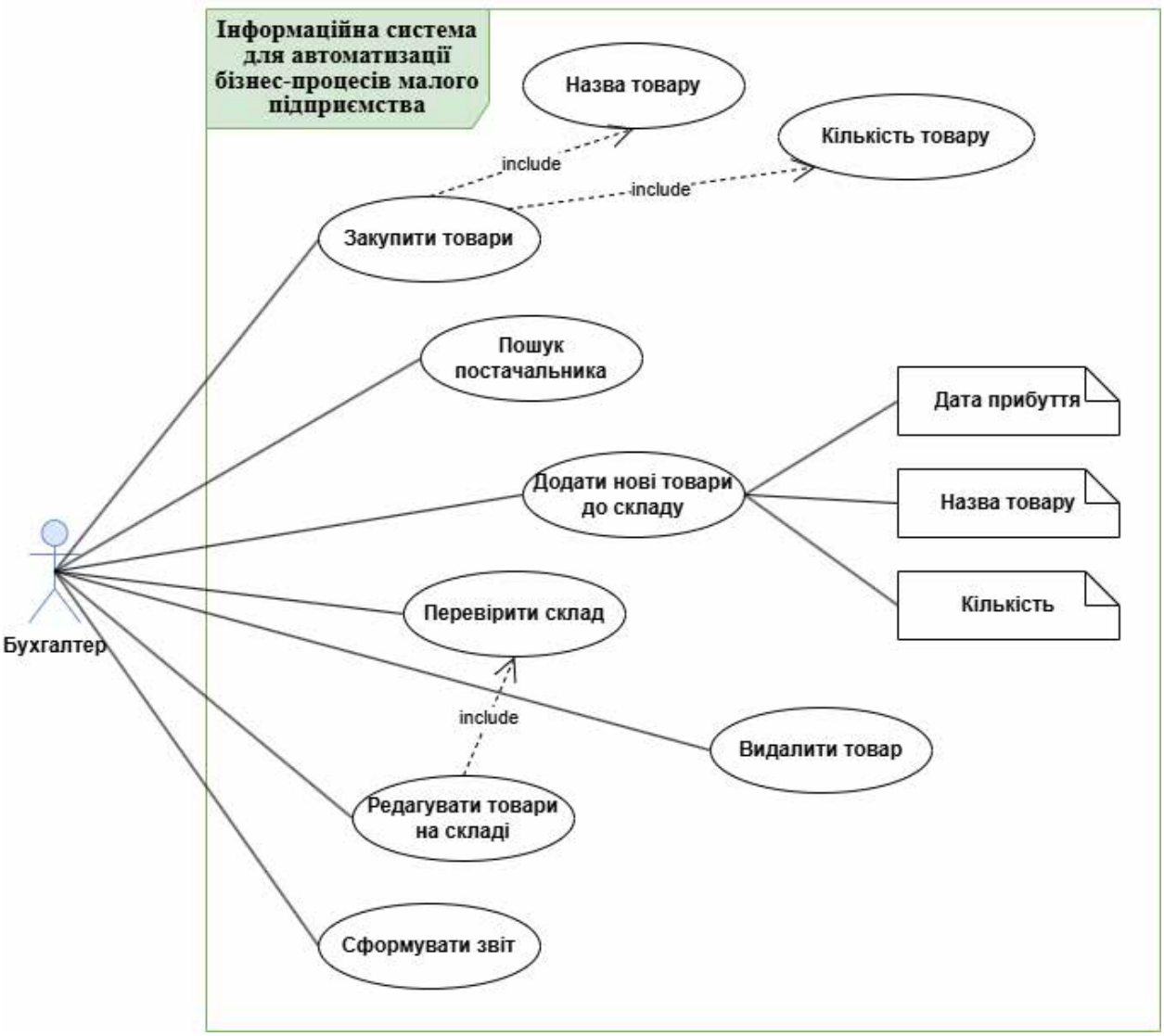

Рис. 2.1 – Діаграма прецедентів

Розглянемо детальніше вищеописані прецеденти.

Сценарій використання: "Закупити товари"

Актор: Бухгалтер

Передумови: Бухгалтер входить в інформаційну систему з відповідними дозволами для управління закупівлями. Доступний список необхідних товарів, включаючи їх характеристики та кількість.

Основний потік:

1. бухгалтер отримує доступ до модуля «Закупівля товарів» в інформаційній системі;

2. система відображає поточні рівні запасів, виділяючи товари, які потрібно поповнити на основі попередньо визначених порогових значень;
3. бухгалтер переглядає список товарів, щоб визначити, які позиції необхідно замовити;
4. для кожної позиції, що закуповується, бухгалтер вибирає товар з інвентарного списку та вказує необхідну кількість;
5. бухгалтер ініціює пошук потенційних постачальників за допомогою функції «Пошук постачальника» в системі;
6. система отримує список затверджених постачальників, які пропонують вибрані товари, а також інформацію про їхні ціни та наявність;
7. бухгалтер оцінює постачальників на основі таких критеріїв, як вартість, час доставки та минулі показники;
8. після вибору відповідного постачальника бухгалтер підтверджує деталі замовлення, включаючи назви продуктів, кількість, інформацію про постачальника та очікувану дату доставки;
9. бухгалтер подає замовлення на закупівлю через систему;
10. система генерує документ замовлення на купівлю та надсилає його обраному постачальнику електронною поштою або через портал постачальника;
11. система оновлює статус запасів і записує замовлення в історію покупок для подальшого використання;
12. бухгалтер отримує підтвердження від постачальника щодо замовлення, яке реєструється в системі;
13. після доставки товарів бухгалтер звіряє отримані предмети з замовленням на купівлю та оновлює інвентар, щоб відобразити нові рівні запасів;

14. у разі виявлення будь-яких розбіжностей (наприклад, відсутність товарів або неправильна кількість), бухгалтер зв’язується з постачальником для вирішення проблеми.

Альтернативні потоки:

1. якщо бухгалтер не може знайти відповідного постачальника, він може вибрати створення нового запису постачальника в системі;
2. у разі виникнення будь-яких проблем під час процесу покупки (наприклад, системні помилки, відсутність постачальника), бухгалтер може скасувати закупівлю та задокументувати причину скасування.

Цей сценарій ілюструє, як бухгалтер взаємодіє з інформаційною системою, щоб ефективно керувати закупівлею товарів, гарантуючи, що підприємство підтримує достатній рівень запасів і працює безперебійно.

Розглянемо сценарій виконання іншого прецеденту.

Сценарій використання: "Створити постачальника"

Актор: Бухгалтер

Передумови: Бухгалтер увійшов в інформаційну систему з необхідними дозволами для створення та керування записами постачальників. Бухгалтер зібрав всю необхідну інформацію про нового постачальника.

Основний потік:

1. бухгалтер переходить до розділу «Управління постачальниками» інформаційної системи;
2. система пропонує опцію «Створити нового постачальника»;
3. бухгалтер вибирає цю опцію, щоб ініціювати процес створення постачальника;
4. система відображає форму, яка вимагає від бухгалтера ввести важливі відомості про постачальника;
5. бухгалтер заповнює форму з відповідною інформацією для нового постачальника;
6. система перевіряє введені дані, щоб переконатися, що всі

необхідні поля заповнені та інформація правильно відформатована;

7. після успішної перевірки бухгалтер надає інформацію про постачальника;
8. система зберігає новий запис про постачальника в базі даних і генерує унікальний ідентифікатор постачальника для використання в майбутньому;
9. система відобразить повідомлення про підтвердження, яке вказує на успішне створення постачальника;
10. бухгалтер має можливість переглядати щойно створений профіль постачальника, де він може переглядати введену інформацію та вносити необхідні зміни.

Альтернативні потоки:

1. якщо під час введення даних бухгалтер стикається з помилкою (наприклад, відсутність обов'язкових полів), система запропонує йому виправити інформацію перед подачею;
2. якщо бухгалтер розуміє, що йому потрібна додаткова інформація про постачальника, він може зберегти чернетку запису про постачальника та заповнити його пізніше.

Цей сценарій ілюструє, як бухгалтер взаємодіє з інформаційною системою для створення та керування записами постачальників, сприяючи безперебійній роботі процесів закупівель на малому підприємстві.

2.1.2 Діаграма послідовності. Діаграма послідовності — це певний тип діаграми взаємодії, який використовується в уніфікованій мові моделювання (UML) для ілюстрації того, як об'єкти спілкуються в певному сценарії використання. Ця діаграма фокусується на послідовності повідомлень, якими обмінюються різні об'єкти або компоненти протягом

певного часу, пропонуючи чітке візуальне представлення динамічної поведінки в системі.

Ключові елементи діаграми послідовності включають акторів, об’єкти, лінії життя, повідомлення та блоки активації. Актори представляють зовнішні сутності, які взаємодіють із системою, наприклад користувачів або інші системи, і зазвичай зображуються у вигляді фігурок у верхній частині діаграми. Об’єкти, які є компонентами або екземплярами, що беруть участь у взаємодії, відображаються у вигляді прямокутних рамок із їхніми іменами, що відображаються у верхній частині.

Лінії життя, зображені у вигляді вертикальних пунктирних ліній, що простягаються вниз від кожного актора чи об’єкта, вказують на їхню присутність у часі та допомагають візуалізувати тривалість участі кожного об’єкта у взаємодії. Повідомлення, представлені горизонтальними стрілками, ілюструють спілкування між акторами та об’єктами або між самими об’єктами. Ці повідомлення позначені діями або функціями, що викликаються, і можуть вказувати на синхронне або асинхронне спілкування. Синхронні повідомлення, зображені суцільними лініями та зафарбованими стрілками, означають, що відправник очікує на відповідь, перш ніж продовжити, тоді як асинхронні повідомлення, зображені відкритими стрілками, вказують на те, що відправник продовжує, не чекаючи відповіді [8].

Поля активації — це прямокутники, розташовані на життєвих лініях, щоб показати тривалість, протягом якої об’єкт активний або обробляє

повідомлення. Ці поля виділяють час, який об’єкт витрачає на виконання дії у відповідь на повідомлення.

Діаграма послідовності, представлена на малюнку 2.2, включає кілька ключових об’єктів:

- "Бухгалтер";
- "Товар";
- "Постачальник";
- "Склад".

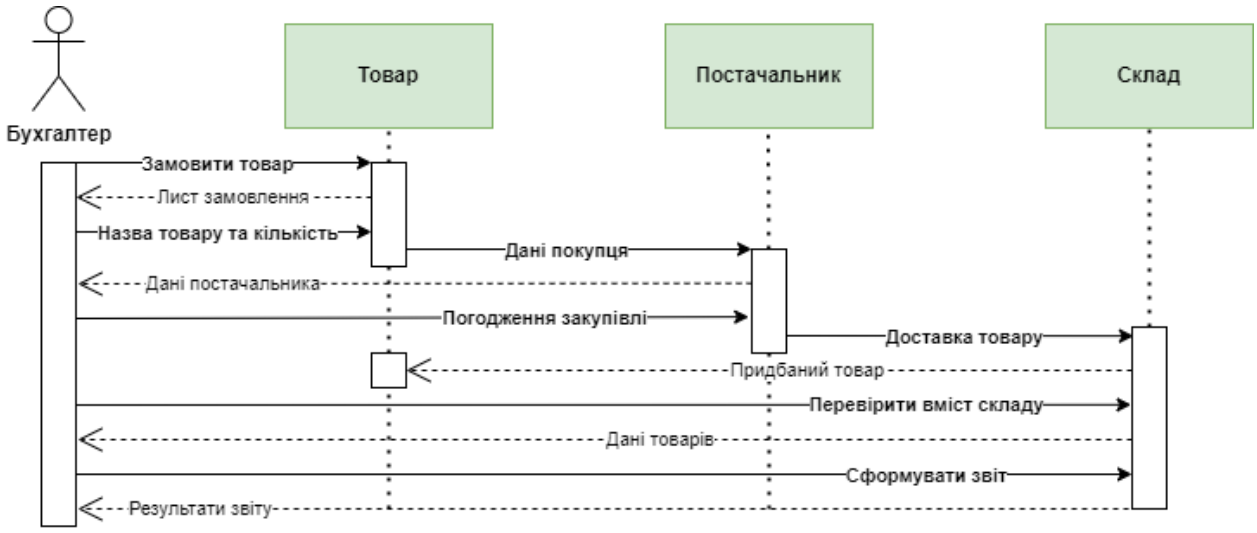

Рис. 2.2 – Діаграма послідовності

У цьому сценарії об’єкт "Бухгалтер" ініціює процес, надсилаючи запит до об’єкта "Товар" для замовлення товару. У відповідь бухгалтер отримує лист замовлення. Після цього він передає назву товару та його кількість, а також дані покупця об’єкту "Постачальник".

Об’єкт "Постачальник" обробляє запит і надсилає свої дані бухгалтеру, після чого "Бухгалтер" підтверджує закупівлю. Наступним етапом є доставка товару від "Постачальника" до "Складу".

Після отримання товару бухгалтер має можливість перевірити вміст складу, отримати інформацію про закуплені товари, а також сформувати звіт і

переглянути його результати. Цей процес ілюструє, як різні об’єкти взаємодіють один з одним у рамках автоматизації бізнес-процесів.

2.1.3 Діаграма активності. Діаграма діяльності — це тип діаграми поведінки в уніфікованій мові моделювання (UML), яка ілюструє потік дій і дій у системі чи процесі. Ця діаграма візуально представляє робочі процеси, висвітлюючи послідовність кроків, необхідних для виконання конкретного завдання або досягнення певної мети. Діаграми діяльності особливо ефективні для моделювання складних процесів, бізнес-процесів і взаємодії між різними компонентами в системі [9].

Ключові елементи діаграми діяльності включають дії, переходи, початкові та кінцеві вузли, вузли прийняття рішень, розгалуження, з’єднання та доріжки. Діяльність представляє окремі завдання або дії, які виконуються в процесі, і зображується як округлені прямокутники. Переходи — це стрілки, які з’єднують ці дії, вказуючи на перехід від одного завдання до іншого та можуть містити умови, які диктують цей потік.

Початковий вузол, представлений заповненим чорним колом, позначає початкову точку робочого процесу, тоді як кінцевий вузол, зображений у вигляді залитого чорного кола з оточуючим кільцем, означає кінець процесу. Вузли прийняття рішень, показані ромбами, вказують точки в робочому процесі, де потрібно зробити вибір, що веде до різних шляхів на основі конкретних умов.

Розгалуження та об’єднання дозволяють відображати одночасні дії в рамках процесу. Розгалуження розбиває потік на кілька паралельних шляхів, тоді як об’єднання об’єднує ці шляхи в один потік [9]. Крім того, доріжки — це горизонтальні або вертикальні поділки, які класифікують дії відповідно до

ролей або компонентів, відповідальних за них, допомагаючи прояснити відповідальність у процесі.

Рис. 2.3 Діаграма активності

2.2 Абстракції предметної області

Абстракції предметної області — це концептуальні моделі, які фіксують основні характеристики та поведінку конкретної сфери інтересів, спрощуючи та пропускаючи непотрібні деталі. Ці абстракції зосереджені на

фундаментальних концепціях, зв’язках і правилах, які визначають певну область, що дозволяє зацікавленим сторонам зосередитися на ключових елементах, пов’язаних з їхніми цілями.

У контексті розробки програмного забезпечення та проєктування системи, абстракції предметної області відіграють вирішальну роль у створенні спільного розуміння серед членів команди. Вони сприяють спілкуванню та співпраці, надаючи огляд домену на високому рівні, керуючи процесом розробки та інформуючи про дизайнерські рішення.

Ключові компоненти абстракцій предметної області включають сутності, зв’язки, атрибути, поведінку та правила. Сутності представляють первинні об’єкти або концепції в межах домену, що мають чітке існування та ідентичність. Приклади сутностей можуть включати клієнтів, продукти або замовлення, кожна з яких інкапсулює власні атрибути та поведінку [10].

Відносини визначають, як ці сутності взаємопов’язані, які можуть варіюватися від «один до одного» до «багато до багатьох», залежно від їх взаємодії в межах домену. Атрибути надають додатковий контекст і інформацію про сутності, такі як сутність «Клієнт», яка може мати такі атрибути, як ім’я, електронна адреса та номер телефону.

Крім того, абстракції предметної області фіксують поведінку або дії, які можуть виконувати сутності, часто виражені як методи або функції. Наприклад, сутність «Продукт» може включати такі дії, як «розрахувати ціну» або «оновити рівень запасів». Крім того, ці абстракції можуть охоплювати конкретні правила або обмеження, що регулюють те, як сутності взаємодіють і поводяться, гарантуючи, що система дотримується визначеної бізнес-логіки.

Застосовуючи абстракції предметної області, розробники можуть створювати більш зручні в обслуговуванні та масштабовані системи. Ці абстракції сприяють розділенню завдань і покращують розуміння основних бізнес-процесів. Крім того, проєктування, кероване доменом (DDD), — це методологія, яка підкреслює важливість моделювання програмного забезпечення на основі конкретних потреб і складності домену. Цей підхід

зрештою призводить до систем, які більше відповідають бізнес-цілям і вимогам користувачів.

Для розробленої системи були розроблені абстракції, як показано на малюнках 2.4 та 2.5.

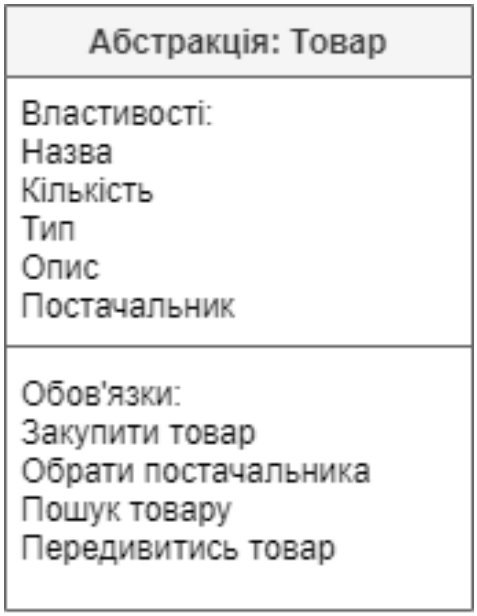

Рис. 2.4 Абстракція «Товар»

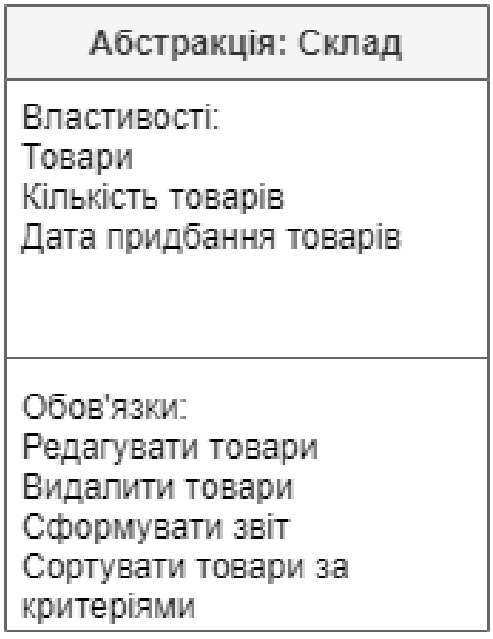

Рис. 2.5 Абстракція «Склад»

2.3 Діаграма класів

Діаграма класів — це найважливіший тип структурної діаграми в уніфікованій мові моделювання (UML), яка візуально представляє класи в системі разом із їхніми атрибутами, методами та зв'язками між ними. Ця діаграма служить схемою для програмних систем, особливо в об'єктно-орієнтованому проєктуванні, і допомагає розробникам і зацікавленим сторонам зрозуміти архітектуру системи та взаємодію компонентів.

В основі діаграми класів знаходяться класи, які зображені у вигляді прямокутників, розділених на три секції. Верхній розділ містить назву класу, середній розділ перераховує атрибути, які визначають характеристики класу, а нижній розділ описує методи, які представляють поведінку або дії, які може виконувати клас. Наприклад, клас «Клієнт» може мати такі атрибути, як «ім'я», «електронна пошта» та «номер телефону», а також такі методи, як «placeOrder()» або «updateProfile()».

Атрибути — це елементи даних, пов'язані з класом, які вказують на його властивості, і можуть мати модифікатори видимості — наприклад, загальнодоступні, приватні або захищені — які визначають їх доступність з інших класів. Методи визначають, як клас взаємодіє з іншими класами, і можуть містити параметри та типи повернення [10].

Діаграми класів також ілюструють різні відносини між класами. Асоціація вказує на зв'язок між двома класами, які можна класифікувати як «один до одного», «один до багатьох» або «багато до багатьох». Агрегація являє собою відношення «ціле-частина», де один клас (ціле) складається з одного або кількох екземплярів іншого класу (частин). Наприклад, клас «Бібліотека» може об'єднувати декілька класів «Книга». Композиція, сильніша форма агрегації, означає залежність життєвого циклу між цілим і його частинами, тобто якщо ціле руйнується, частини також усуваються. Прикладом може бути клас «Будинок», що складається з класів «Кімната». Успадкування дозволяє одному класу (дочірньому або підкласу) успадковувати атрибути та методи від іншого класу

(батьківського або суперкласу), сприяючи повторному використанню коду та ієрархічним структурам.

Діаграми класів надають численні переваги, включаючи чітке та організоване представлення структури системи. Ця ясність допомагає зрозуміти та передавати дизайнерські рішення членам команди та зацікавленим сторонам, допомагаючи виявити потенційні проблеми дизайну та багаторазові компоненти. Крім того, діаграми класів служать важливою документацією для системи, полегшуючи подальше обслуговування та покращення, водночас керуючи впровадженням класів та їх взаємодією в кодовій базі.

Спеціальна діаграма класів була створена з використанням об’єктно-орієнтованого підходу для конкретної веб-програми (рис. 2.6).

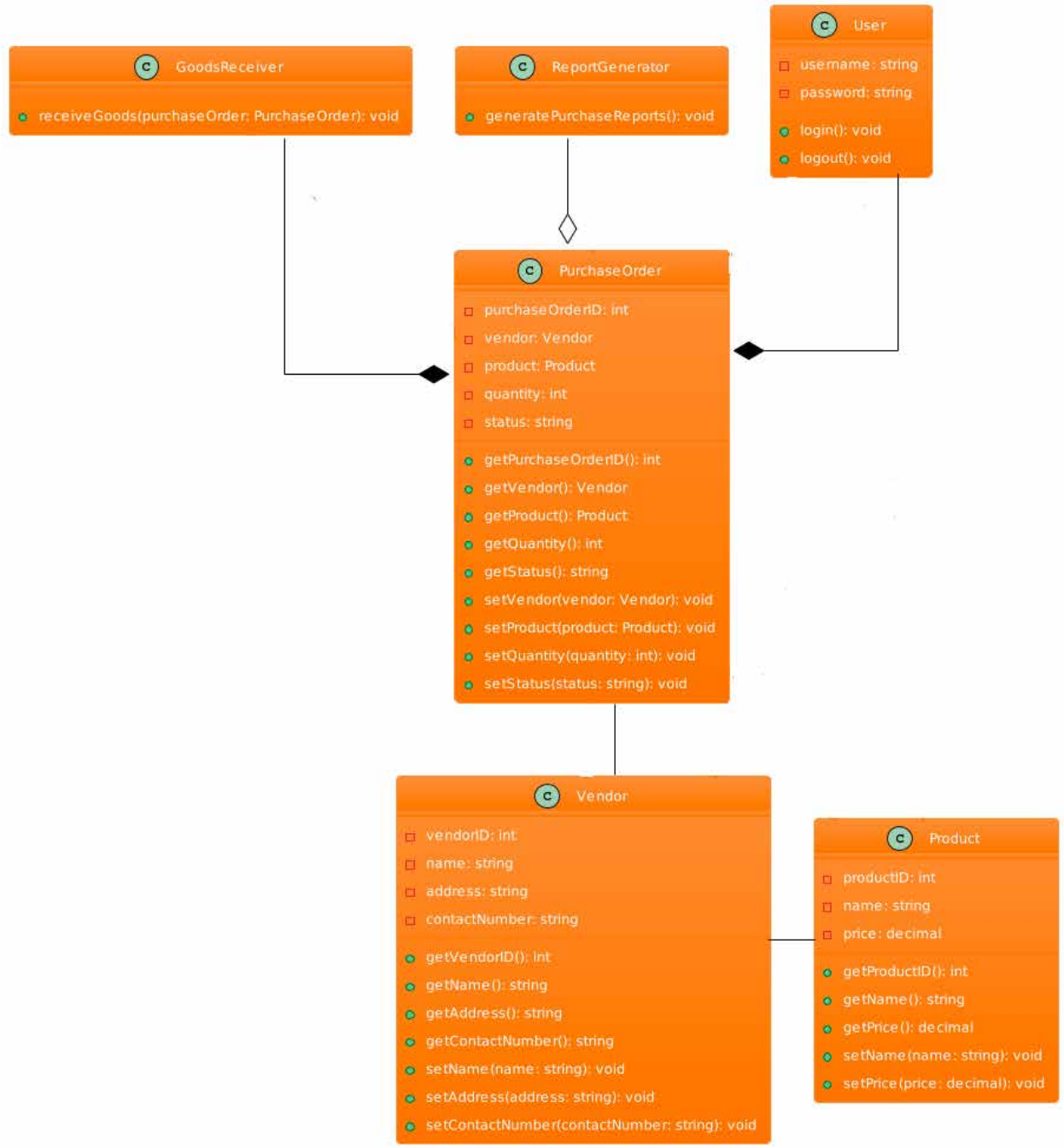

Рис. 2.6 Діаграма класів

Діаграма класів для системи, зосередженої на зовнішньому вигляді та аналізі закупівельної діяльності компанії, охоплює різні класи та їхні зв’язки, усі спрямовані на забезпечення безпеки програмного забезпечення. Ця діаграма візуально представляє основні компоненти системи та ілюструє, як вони взаємодіють один з одним.

Одним із ключових класів є «PurchaseOrder», який інкапсулює окремі закупівельні дії в організації. Цей клас включає такі атрибути, як ідентифікатор

покупки, дата та загальна сума. Крім того, він може містити методи для виконання перевірок і розрахунків, пов’язаних із покупками, покращуючи функціональність процесу купівлі.

Іншим важливим класом є «Користувач», який представляє осіб, які взаємодіють із системою. Він містить ключові атрибути, такі як ім’я користувача, пароль і роль, які необхідні для підтримки заходів безпеки. Клас також може включати методи автентифікації користувача та контролю доступу, гарантуючи, що лише авторизований персонал може виконувати певні дії.

Клас "Продукт" означає товари або послуги, залучені в купівельну діяльність. Він містить такі атрибути, як ідентифікатор продукту, назва, опис і ціна. Крім того, цей клас може містити пов’язані з безпекою атрибути, такі як дозволи на доступ або механізми шифрування, для захисту конфіденційної інформації.

Діаграма класів також висвітлює зв'язки між цими різними класами. Наприклад, може існувати зв’язок між класом «Постачальник» і класом «Продукт», що вказує на те, що постачальник може пропонувати кілька продуктів. Подібним чином зв’язки між класом «PurchaseOrder» і класом «User» представляють користувача, відповідального за конкретну покупку, тим самим пов’язуючи діяльність із закупівлі з конкретними особами.

2.4 Логічна модель даних

ERwin — це потужний інструмент моделювання даних, який широко використовується для проєктування, візуалізації та керування моделями даних і базами даних. Він надає комплексне середовище для адміністраторів баз даних, архітекторів даних і розробників для створення та підтримки логічних і фізичних моделей даних, полегшуючи ефективне керування даними та забезпечуючи цілісність даних.

Однією з головних особливостей ERwin є його здатність створювати діаграми сутності та зв’язку (ER), які візуально представляють структуру бази

даних. Ці діаграми ілюструють сутності, їхні атрибути та зв’язки між ними, дозволяючи користувачам зрозуміти потік даних і організацію даних у системі. Це візуальне представлення допомагає в процесі проєктування, дозволяючи зацікавленим сторонам швидко зрозуміти складні структури даних і зв’язки [11].

ERwin підтримує як логічне, так і фізичне моделювання, дозволяючи користувачам визначати моделі даних на різних рівнях абстракції. У логічному моделюванні користувачі можуть зосередитися на концептуальному представленні даних, визначенні сутностей і зв’язків, не турбуючись про те, як дані будуть фізично зберігатися. Навпаки, фізичне моделювання передбачає визначення того, як дані будуть реалізовані в конкретній системі керування базами даних (СУБД), включаючи такі деталі, як типи даних, індекси та параметри зберігання.

Інструмент також пропонує функції для зворотного проєктування, що дозволяє користувачам генерувати моделі даних з існуючих баз даних. Ця функція особливо корисна для розуміння застарілих систем, оскільки вона дозволяє користувачам візуалізувати та документувати поточний стан бази даних, полегшуючи планування оновлень або міграцій.

ERwin сприяє співпраці між членами команди, дозволяючи кільком користувачам працювати над однією моделлю даних одночасно. Він забезпечує контроль версій і функції керування змінами, гарантуючи, що всі зміни відстежуються та що користувачі можуть легко повернутися до попередніх версій, якщо це необхідно.

Крім того, ERwin інтегрується з різними платформами баз даних, такими як Oracle, SQL Server, MySQL і PostgreSQL, що дозволяє користувачам генерувати сценарії SQL для створення або зміни структур бази даних безпосередньо зі своїх моделей. Ця повна інтеграція оптимізує процес розробки та зменшує ризик помилок під час впровадження [11].

Підсумовуючи, ERwin — це надійний інструмент моделювання даних, який покращує дизайн та керування базами даних за допомогою візуального представлення, можливостей логічного та фізичного моделювання, зворотного

проєктування та функцій співпраці. Надаючи комплексне середовище для моделювання даних, ERwin допомагає організаціям підтримувати цілісність даних, оптимізувати продуктивність бази даних і підтримувати ефективні практики управління даними.

Діаграма «сутність-зв'язок» (ER) — це візуальне представлення, яке зображує зв'язки між сутностями в базі даних. Ця діаграма має важливе значення для розробки бази даних, оскільки вона ілюструє структуру даних і взаємодію між різними об'єктами даних у системі. Діаграми ER, які в основному використовуються на етапі проєктування, допомагають розробникам і зацікавленим сторонам зрозуміти вимоги до даних і зв'язки перед фактичним впровадженням.

Основні компоненти діаграми ER включають сутності, атрибути, зв'язки та потужність. Сутності представляють різні об'єкти або поняття в базі даних, наприклад «Клієнт», «Продукт» або «Замовлення». Ці сутності зазвичай відображаються на діаграмі у вигляді прямокутників. Атрибути надають додаткову інформацію про кожну сутність, визначаючи її властивості або характеристики. Наприклад, сутність «Клієнт» може мати такі атрибути, як «Ідентифікатор клієнта», «Ім'я», «Електронна адреса» та «Номер телефону», які зображені у вигляді овалів, з'єднаних із відповідними сутностями.

Зв'язки на діаграмі ER ілюструють, як сутності пов'язані одна з одною. Вони представляють асоціації та можуть відрізнятися за типом, включаючи зв'язки «один до одного», «один до багатьох» або «багато до багатьох». Відносини зазвичай представлені у вигляді ромбів із лініями, що з'єднують їх із пов'язаними сутностями. Кардинальність визначає числові зв'язки між сутностями у зв'язку, вказуючи, скільки екземплярів однієї сутності можуть бути пов'язані з екземплярами іншої. Наприклад, клієнт може мати кілька замовлень, що ілюструє зв'язок «один до багатьох».

Діаграми ER пропонують численні переваги. Вони забезпечують чітке та стисле візуальне представлення структури даних, що полегшує зацікавленим сторонам розуміння вимог до системи. Відображаючи сутності та їхні зв'язки, ці

діаграми допомагають виявити потенційні проблеми дизайну, гарантуючи, що база даних може ефективно підтримувати необхідні операції та запити.

Крім того, ER-діаграми служать схемою для створення баз даних, керуючи розробниками в реалізації схеми бази даних. Вони також важливі для документування та спілкування, допомагаючи гарантувати, що всі, хто бере участь у проєкті, мають спільне розуміння моделі даних.

Створена логічна модель інформаційного забезпечення відповідатиме цим вимогам, забезпечуючи наочне та зручне представлення інформаційної системи.

Логічна модель системи, що розвивається (рис. 2.7), складається з наступних сутностей:

- user;
- customer;
- supplier;
- product;
- purchase_request;
- sale_request.

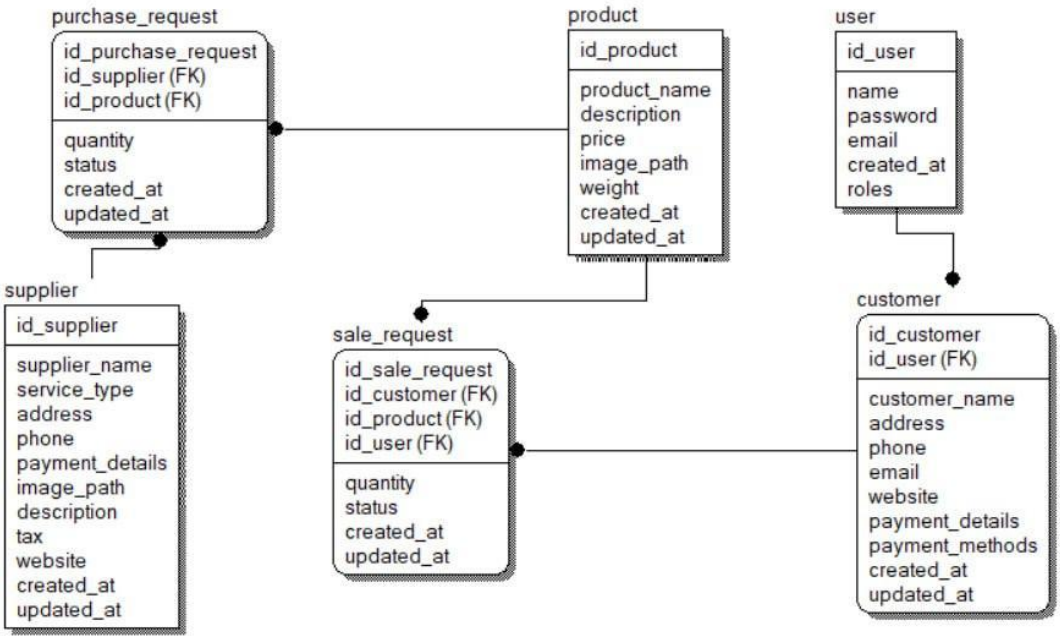

Рис. 2.7 ER-діаграма

Сутність «Product» пов’язана з сутностями «Purchase_Request» та «Sale_Request». Також сутність «Purchase» зв’язана з сутностями «Product» та «Supplier». І сутність “Customer” зв’язана з сутностями “User” та “Sale_Request”.

2.5 Висновки до 2 розділу

У другому розділі виконано комплексне моделювання предметної області, що стало основою для подальшої розробки інформаційної системи. Спочатку було проведено об’єктне та функціональне моделювання, що дозволило виділити ключові сутності, їх властивості та взаємозв’язки, а також основні бізнес-процеси, які має підтримувати система.

Далі були сформовані абстракції предметної області, що забезпечило узагальнення і спрощення складних елементів системи без втрати їх суттєвих характеристик. Це дозволило краще структурувати інформацію і забезпечити логічну цілісність майбутньої реалізації.

На основі отриманих даних побудовано діаграму класів, яка відображає структуру системи та взаємодію основних об’єктів, що забезпечує чітке уявлення про архітектуру та логіку роботи системи.

Завершальним етапом моделювання стала розробка логічної моделі даних, що враховує всі необхідні зв’язки між сутностями та оптимізує збереження інформації у базі даних.

Отже, виконані моделювальні роботи створили міцну концептуальну базу для проєктування програмної системи, що відповідає поставленим функціональним вимогам і забезпечує ефективну автоматизацію бізнес-процесів малого підприємства.

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Вибір системи управління базою даних та її реалізація

Реляційні бази даних — це тип системи керування базами даних (СУБД), яка зберігає дані в структурованому форматі за допомогою рядків і стовпців у таблицях. Така організація забезпечує ефективний пошук, маніпулювання та керування даними. Розроблені на основі реляційної моделі, запропонованої Е. Ф. Коддом у 1970-х роках, реляційні бази даних стали стандартом для зберігання даних у різних програмах, від малих підприємств до великих корпоративних систем [12].

В основі реляційної бази даних лежить концепція таблиць, де кожна таблиця представляє певну сутність, наприклад клієнтів, замовлення або продукти. Кожна таблиця складається з рядків (також відомих як записи або кортежі) і стовпців (також відомих як поля або атрибути). Кожен рядок відповідає унікальному екземпляру сутності, тоді як кожен стовпець представляє певний атрибут цієї сутності. Наприклад, таблиця «Клієнт» може містити такі стовпці, як «Ідентифікатор клієнта», «Ім’я», «Електронна адреса» та «Номер телефону».

Однією з ключових особливостей реляційних баз даних є використання первинних ключів і зовнішніх ключів для встановлення зв’язків між таблицями. Первинний ключ — це унікальний ідентифікатор для кожного запису в таблиці, що забезпечує відсутність двох записів з однаковим значенням цього ключа. Зовнішні ключі — це атрибути в одній таблиці, які посилаються на первинний ключ іншої таблиці, створюючи зв’язок між двома таблицями. Цей зв’язок дозволяє представляти складні структури даних і забезпечує ефективне надсилання запитів до кількох таблиць.

Реляційні бази даних використовують мову структурованих запитів (SQL) як стандартну мову для визначення та обробки даних. SQL надає потужні можливості для запиту даних, вставки нових записів, оновлення існуючих записів і видалення записів. Він також підтримує складні операції, такі як об’єднання, які дозволяють користувачам об’єднувати дані з кількох таблиць на основі визначених зв’язків.

Однією з значних переваг реляційних баз даних є їх здатність підтримувати цілісність і узгодженість даних за допомогою обмежень і транзакцій. Обмеження, такі як унікальні обмеження та обмеження посилальної цілісності, допомагають гарантувати, що дані дотримуються певних правил і зв’язків. Транзакції дають змогу виконувати декілька операцій як єдине ціле, гарантуючи, що або всі операції будуть завершені успішно, або жодної, що допомагає підтримувати узгодженість даних навіть у разі помилок або збоїв системи.

Реляційні бази даних широко використовуються в різних галузях завдяки своїй надійності, гнучкості та простоті використання. Вони підходять для додатків, які вимагають структурованого зберігання даних і складних запитів, що робить їх ідеальними для систем управління взаємовідносинами з клієнтами (CRM), систем планування ресурсів підприємства (ERP), платформ електронної комерції та багатьох інших програм, керованих даними.

Таким чином, реляційні бази даних є фундаментальним компонентом сучасного управління даними, що забезпечує структурований підхід до зберігання, пошуку та маніпулювання даними. Використання в них таблиць, зв’язків і SQL забезпечує ефективну обробку даних і цілісність, що робить їх кращим вибором для багатьох організацій, які прагнуть ефективно керувати своїми даними [12].

Рішення вибрати MySQL для розробки інформаційної системи, спрямованої на автоматизацію бізнес-процесів для малого підприємства, було зумовлене кількома вагомими факторами. MySQL — це широко визнана система керування реляційною базою даних (RDBMS), яка пропонує численні переваги, що робить її ідеальним вибором для цього проєкту.

Однією з основних причин вибору MySQL є його природа з відкритим вихідним кодом, що дозволяє економічно ефективне впровадження без ліцензійних зборів. Це особливо корисно для малих підприємств, які можуть мати обмежений бюджет, оскільки дає їм змогу використовувати надійне рішення бази даних без значних витрат.

MySQL відома своєю надійністю та стабільністю, які є вирішальними для будь-якої бізнес-системи, яка потребує послідовного керування даними та доступності. Його міцна репутація в галузі гарантує, що він може відповідати вимогам зростаючого бізнесу, забезпечуючи необхідну підтримку цілісності даних і продуктивності.

Крім того, MySQL пропонує чудову продуктивність і швидкість у виконанні запитів, що важливо для інформаційної системи, яка оброблятиме різні бізнес-процеси. Його здатність ефективно керувати великими обсягами даних дозволяє користувачам швидко отримувати та маніпулювати інформацією, підвищуючи загальну швидкість реагування системи.

Сумісність MySQL з різними мовами програмування та платформами додатково підтримує його вибір для цього проєкту. Оскільки інформаційна

система розробляється з використанням PHP і Symfony, MySQL бездоганно інтегрується з цими технологіями, забезпечуючи ефективну взаємодію та маніпулювання даними. Ця сумісність спрощує процес розробки та зменшує потенційні проблеми інтеграції.

Крім того, MySQL забезпечує надійні функції безпеки, необхідні для захисту конфіденційних бізнес-даних. Завдяки підтримці автентифікації користувачів, контролю доступу та шифрування даних MySQL гарантує, що інформація, що зберігається в базі даних, залишається безпечною та доступною лише авторизованому персоналу.

Активна спільнота MySQL і обширна документація пропонують цінні ресурси для розробників. Ця мережа підтримки надає доступ до форумів, навчальних посібників і найкращих практик, що полегшує команді розробників усунення неполадок і вдосконалює свої навички під час роботи над проєктом.

Під час проєктування таблиць у фізичній моделі основою слугували сутності з логічної моделі. Атрибути цих сутностей були використані для розробки структури таблиці. Отриману схему бази даних зображено на рис. 10.

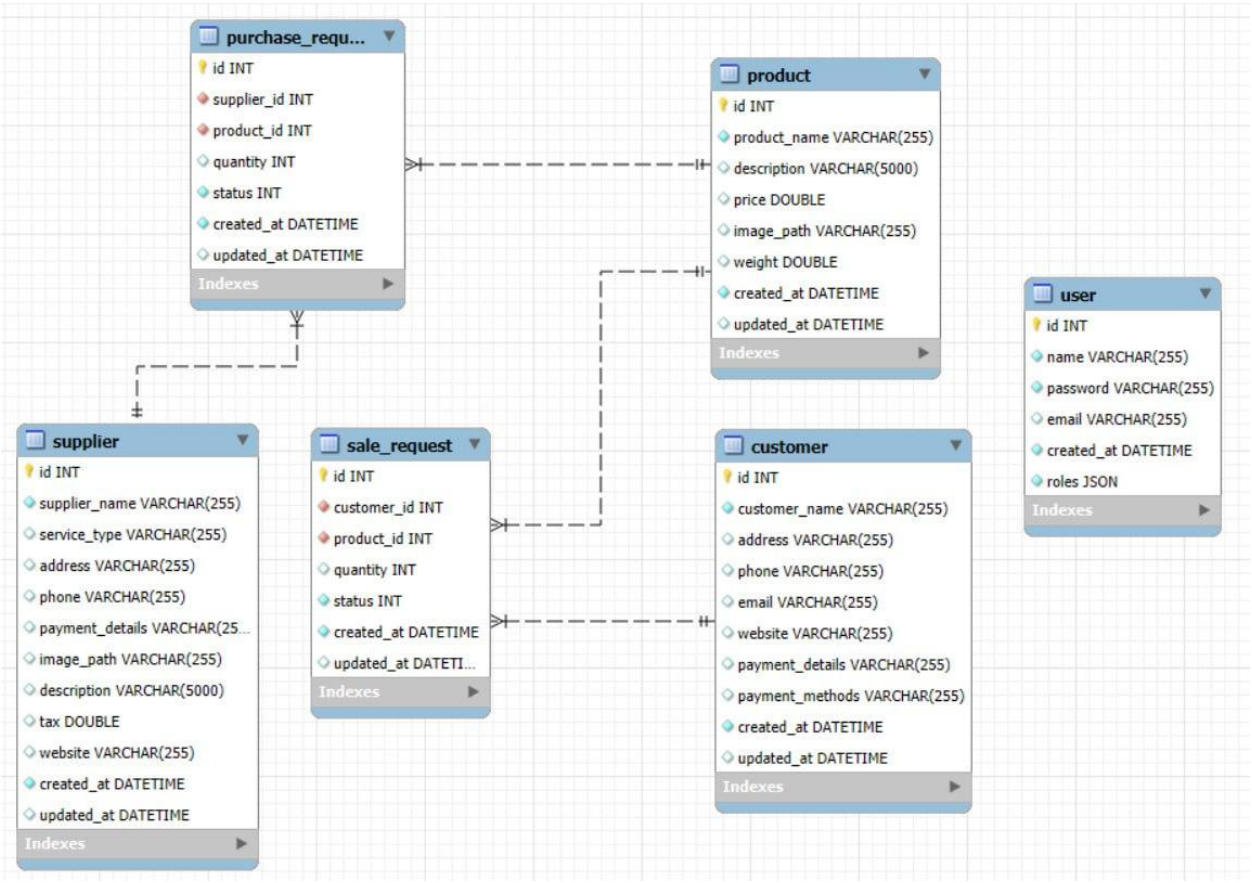

Рис. 3.1 База даних системи

База даних для інформаційної системи автоматизації бізнес-процесів малого підприємства складається з наступних таблиць: User, Supplier, Customer, Product, PurchaseRequest, SaleRequest. Кожна таблиця відповідає сутності в системі та містить певні поля, які визначають атрибути цих сутностей. Зв’язки між таблицями встановлюють взаємозв’язок даних, сприяючи ефективному управлінню та пошуку даних.

User:

1. id: ціле число, первинний ключ, автоматично згенерований;
2. ім'я: рядок (255 символів), ім'я користувача;
3. пароль: рядок (255 символів), пароль користувача;
4. email: Рядок (255 символів, nullable), адреса електронної пошти користувача;
5. createdAt: DateTimeImmutable, дата створення користувача;
6. roles: масив ролей, призначених користувачеві.

Supplier:

1. id: ціле число, первинний ключ, автоматично згенерований;
2. providerName: рядок (255 символів), назва постачальника;
3. serviceType: рядок (255 символів, значення null), тип послуг, що надаються;
4. адреса: Рядок (255 символів, nullable), адреса постачальника;
5. phone: рядок (255 символів, nullable), номер телефону постачальника;
6. paymentDetails: Рядок (255 символів, nullable), платіжна інформація;
7. imagePath: рядок (255 символів, значення null), шлях до зображення постачальника;
8. description: Рядок (5000 символів, nullable), опис постачальника;
9. tax: Float, податкова інформація для постачальника;
10. веб-сайт: рядок (255 символів, значення null), веб-сайт постачальника;
11. createdAt: DateTimeImmutable, дата створення постачальника;
12. updatedAt: DateTimeImmutable (nullable), дата останнього оновлення інформації про постачальника.

Customer:

1. id: ціле число, первинний ключ, автоматично згенерований;
2. customerName: рядок (255 символів), ім'я клієнта;
3. адреса: Рядок (255 символів, nullable), адреса клієнта;
4. phone: рядок (255 символів, nullable), номер телефону клієнта;
5. email: Рядок (255 символів, nullable), адреса електронної пошти клієнта;
6. веб-сайт: рядок (255 символів, значення null), веб-сайт клієнта;
7. paymentDetails: Рядок (255 символів, nullable), платіжна інформація;

8. paymentMethods: рядок (255 символів, nullable), прийнятні способи оплати;
9. createdAt: DateTimeImmutable, дата створення клієнта;
10. updatedAt: DateTimeImmutable (nullable), дата останнього оновлення інформації про клієнта.

Product:

1. id: ціле число, первинний ключ, автоматично згенерований;
2. productName: рядок (255 символів), назва продукту;
3. description: Рядок (5000 символів, nullable), опис продукту;
4. price: плаваюча (nullable), ціна товару;
5. imagePath: рядок (255 символів, значення null), шлях до зображення продукту;
6. weight: Float (nullable), вага продукту;
7. createdAt: DateTimeImmutable, дата створення продукту;
8. updatedAt: DateTimeImmutable (nullable), дата останнього оновлення інформації про продукт.

PurchaseRequest:

1. id: ціле число, первинний ключ, автоматично згенерований;
2. постачальник: зв’язок із таблицею постачальника (багато до одного), що вказує постачальника для закупівлі;
3. product: зв’язок із таблицею Product (багато до одного), що вказує продукт, який купується;
4. quantity: ціле число (може null), кількість запитаного продукту;
5. статус: Ціле число, статус заявки на купівлю;
6. createdAt: DateTimeImmutable, дата створення запиту на купівлю;
7. updatedAt: DateTimeImmutable (nullable), дата останнього оновлення запиту на купівлю.

SaleRequest:

1. id: ціле число, первинний ключ, автоматично згенерований;
2. клієнт: зв’язок із таблицею «Клієнт» (багато до одного), що вказує клієнта, який здійснює покупку;
3. product: зв’язок із таблицею Product (багато до одного), що вказує на продукт, що продається;
4. quantity: ціле число (може обнуляти), кількість продукту, який запитується на продаж;
5. статус: Ціле число, статус заявки на продаж;
6. createdAt: DateTimeImmutable, дата створення запиту на продаж;
7. updatedAt: DateTimeImmutable (nullable), дата останнього оновлення запиту на продаж.

Таблиця «User» працює незалежно, оскільки в основному керує автентифікацією користувачів і ролями. Таблиця Supplier пов’язана з таблицею PurchaseRequest за допомогою зв’язку «багато до одного», що вказує на те, що кожен запит на закупівлю пов’язаний з одним постачальником, тоді як постачальник може виконувати кілька запитів на закупівлю. Таблиця Product пов’язана з таблицями PurchaseRequest і SaleRequest. У кожному випадку зв’язок є багато-до-одного, тобто кілька запитів на купівлю чи продаж можуть посилатися на той самий продукт. Таблиця Customer пов’язана з таблицею SaleRequest за допомогою зв’язку «багато до одного», що вказує на те, що клієнт може мати кілька запитів на продаж.

3.2 Архітектура програмного забезпечення

Архітектура програмного забезпечення відноситься до фундаментальних структур і високорівневого дизайну програмної системи, що охоплює організацію її компонентів і їх взаємодію. Він служить схемою як для розробки, так і для підтримки програмного додатку, керуючи критичними рішеннями, пов’язаними з проєктуванням, впровадженням і вибором технології.

За своєю суттю архітектура програмного забезпечення включає різні компоненти, якими можуть бути класи, бібліотеки, служби або API, кожен з яких має певні обов’язки. Ці компоненти взаємодіють один з одним, і розуміння їх взаємозв’язків має важливе значення для забезпечення належного функціонування системи, зберігаючи слабкий зв’язок, де це необхідно [15].

Архітектурні шаблони та шаблони проєктування часто використовуються для вирішення типових проблем проєктування в архітектурі програмного забезпечення. Приклади цих шаблонів включають мікросервіси, архітектуру клієнт-сервер, багаторівневу архітектуру та архітектуру, керовану подіями. Використовуючи встановлені шаблони, розробники можуть оптимізувати процес проєктування та використовувати перевірені рішення.

Атрибути якості відіграють важливу роль у формуванні архітектури програмного забезпечення, впливаючи на такі аспекти, як продуктивність, масштабованість, надійність, безпека та зручність обслуговування. Ці атрибути керують архітектурними рішеннями, щоб забезпечити відповідність системи як функціональним, так і нефункціональним вимогам.

Ефективна архітектура програмного забезпечення супроводжується ретельною документацією, яка описує компоненти, зв’язки, шаблони та атрибути якості. Ця документація має життєво важливе значення для донесення архітектури до зацікавлених сторін, включаючи розробників, керівників проєктів і клієнтів, гарантуючи, що всі учасники мають спільне розуміння дизайну системи.

Крім того, архітектура програмного забезпечення не є статичною; він повинен з часом адаптуватися до мінливих вимог, нових технологій і бізнес-цілей, що розвиваються. Добре розроблена архітектура сприяє гнучкості та адаптивності, полегшуючи внесення змін без значних переробок.

На рішення прийняти мікросервісну архітектуру для інформаційної системи, призначеної для автоматизації бізнес-процесів малого підприємства, вплинуло кілька ключових факторів, які відповідають цілям і вимогам проєкту.

Однією з основних причин вибору мікросервісів є можливість підвищити гнучкість і масштабованість. В архітектурі мікросервісів програма складається з менших незалежних служб, які можна розробляти, розгортати та масштабувати незалежно. Цей модульний підхід дозволяє підприємству швидко реагувати на мінливі потреби бізнесу, дозволяючи додавати нові функції або модифікувати існуючі функції, не впливаючи на всю систему. У міру зростання підприємство може масштабувати окремі послуги відповідно до попиту, оптимізуючи використання ресурсів і продуктивність.

Мікросервіси також сприяють кращій організації та розподілу завдань. Кожен мікросервіс відповідає за конкретну бізнес-можливість або функцію, наприклад керування користувачами, каталог продуктів або обробку замовлень. Такий чіткий розподіл дозволяє командам розробників зосередитися на конкретних областях програми, що сприяє покращенню обслуговування коду та полегшенню усунення несправностей. Крім того, завдяки відокремленню служб команди можуть одночасно працювати над різними аспектами системи, що покращує співпрацю та прискорює процес розробки [16].

Ще однією значною перевагою мікросервісної архітектури є її підтримка різноманітних технологічних стеків. Кожен мікросервіс можна створити за допомогою мови програмування, фреймворку або технології баз даних, які найбільше підходять для його конкретної функції. Ця гнучкість дозволяє команді розробників використовувати найновіші технології та передовий досвід, підвищуючи загальну якість системи. Крім того, це дозволяє підприємству поступово впроваджувати нові технології, мінімізуючи збої під час оновлень.

Мікросервіси також сприяють покращенню ізоляції несправностей і стійкості. У традиційній монолітній архітектурі збій в одній частині програми може призвести до повного виходу з ладу системи. Однак в архітектурі мікросервісу збої містяться в окремих службах, що дозволяє решті програми продовжувати працювати. Ця стійкість має вирішальне значення для підтримки бізнес-операцій і забезпечення позитивної взаємодії з користувачем.

Підсумовуючи, вибір архітектури мікросервісу для інформаційної системи, спрямованої на автоматизацію бізнес-процесів на малому підприємстві, був зумовлений прагненням до гнучкості, масштабованості, покращеної зручності обслуговування, різноманітності технологій, ізоляції помилок і узгодження з практиками DevOps. Такий архітектурний підхід підтримує зростання та адаптивність підприємства, гарантуючи, що система може розвиватися разом із мінливими вимогами бізнесу.

3.3 Організаційна структура програмного забезпечення

3.3.1 Діаграма пакетів. Діаграма пакета — це структурна діаграма на уніфікованій мові моделювання (UML), яка ілюструє, як пакети організовані в системі програмного забезпечення та зв’язки між ними. Пакети служать контейнерами для пов’язаних елементів, таких як класи, інтерфейси та компоненти, що дозволяє краще керувати та організовувати складні системи. Надаючи високорівневе уявлення про архітектуру системи, діаграми пакетів допомагають прояснити взаємодію та залежності між різними частинами програмного забезпечення.

На діаграмі пакетів пакети представлені у вигляді прямокутників із невеликим виступом або кутом. Ці пакети групують пов’язані елементи, полегшуючи розуміння структури системи. У кожен пакет можна включити різні елементи, такі як класи та інтерфейси, демонструючи, які компоненти належать до яких пакетів.

Залежності між пакетами зображено пунктирними стрілками, що вказує на те, що один пакет залежить від іншого. Це представлення допомагає визначити, як зміни в одному пакеті можуть вплинути на інші, що робить його ключовим для управління відносинами в системі. Крім того, діаграми пакетів можуть передавати видимість елементів, таких як публічний, приватний або захищений рівні доступу, вказуючи, як компоненти можуть взаємодіяти в різних пакетах [17].

Переваги використання пакетних діаграм значні. Вони спрощують візуалізацію складних систем, представляючи чіткий огляд організації пакетів та їх зв’язків. Ця чіткість покращує спілкування між членами команди та зацікавленими сторонами, полегшуючи спільне розуміння архітектури системи.

Крім того, пакетні діаграми допомагають ідентифікувати залежності між компонентами, що важливо для керування змінами та підтримки цілісності системи. Логічно організовуючи пов’язані елементи в пакети, розробники можуть просувати модульний дизайн, що полегшує керування, тестування та обслуговування програмного забезпечення з часом.

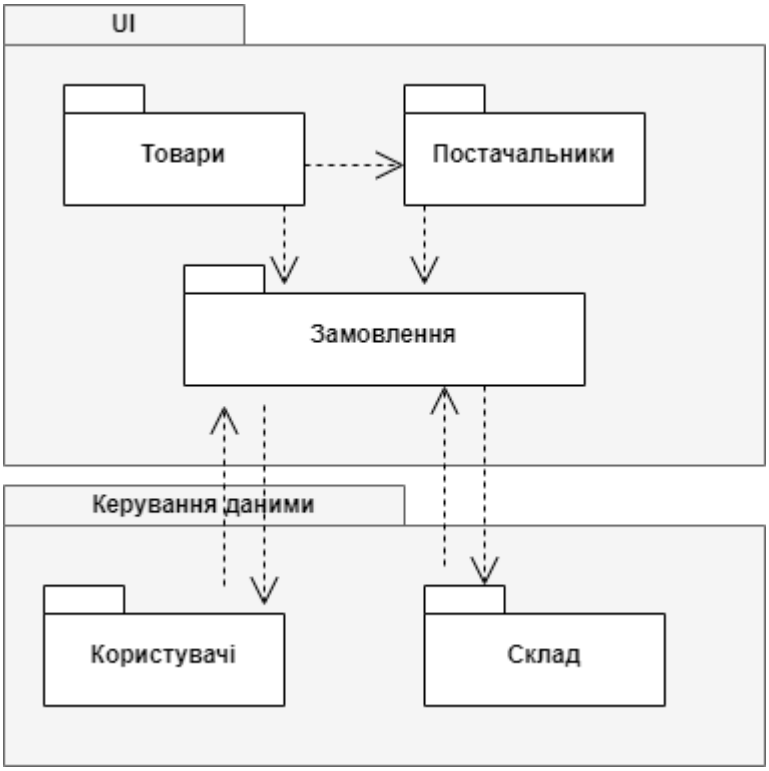

Рис. 3.2 Діаграма пакетів

3.4 Вимоги до апаратного та програмного забезпечення

Успішне впровадження інформаційної системи автоматизації бізнес-процесів на малому підприємстві залежить від специфічних вимог до програмного та апаратного забезпечення. Ці специфікації є важливими для забезпечення ефективної роботи системи, забезпечуючи користувачам надійне середовище для ефективного керування своєю бізнес-діяльністю.

Що стосується програмного забезпечення, то на сервері, на якому розміщено програму, має працювати сумісна операційна система, наприклад Linux (наприклад, Ubuntu або CentOS) або Windows Server. Для розміщення програми та обслуговування веб-сторінок потрібен веб-сервер, наприклад HTTP-

сервер Apache або Nginx. Мова програмування PHP, зокрема версія 7.4 або вище, буде використовуватися для сценаріїв на стороні сервера та логіки додатків. Фреймворк Symfony версії 5.x або новішої забезпечить структуроване середовище та повторно використовувані компоненти для розробки програм.

Щоб полегшити взаємодію з базою даних, Doctrine ORM буде використано для об'єктно-реляційного відображення. Система використовуватиме MySQL як систему керування базою даних для зберігання та пошуку даних. Інтерфейсні технології, включаючи HTML, CSS і Bootstrap 5, будуть використані для створення адаптивного та зручного інтерфейсу.

Важливе значення також має відповідне середовище розробки. Інтегроване середовище розробки (IDE) або редактор коду, наприклад PhpStorm, Visual Studio Code або Sublime Text, допоможуть у написанні та управлінні кодом, тоді як Composer використовуватиметься для керування залежностями в PHP. Git слугуватиме системою контролю версій, забезпечуючи спільну розробку та відстеження змін.

Додаткові інструменти включатимуть веб-браузер, наприклад Chrome або Firefox, для тестування та доступу до програми, а також інструмент тестування API, як-от Postman, для полегшення тестування кінцевих точок API під час розробки.

З боку апаратного забезпечення сервер, на якому розміщено програму, повинен мати відповідні характеристики. Для ефективної обробки одночасних запитів користувачів рекомендується використовувати багатоядерний процесор, наприклад Intel Xeon або AMD Ryzen. Принаймні 8 ГБ оперативної пам'яті необхідно для безперебійної роботи, особливо під час керування кількома запитами та взаємодією з базою даних. Об'єм пам'яті має бути достатнім, в ідеалі 100 ГБ або більше, для розміщення операційної системи, файлів додатків і бази даних, причому для швидшого доступу до даних і продуктивності краще використовувати SSD. Надійне підключення до Інтернету з достатньою пропускною здатністю також має вирішальне значення для підтримки доступу користувачів і передачі даних.

Робочі станції користувача повинні мати власний набір вимог до обладнання. Для роботи веб-браузерів та інших програм рекомендується використовувати двоядерний процесор або вище, а також принаймні 4 ГБ оперативної пам’яті, щоб забезпечити плавну багатозадачність. Робочі станції також повинні забезпечувати достатньо місця для зберігання файлів користувача та мати сумісність із сучасними операційними системами, такими як Windows, macOS або Linux. Остання версія веб-браузера буде необхідною для доступу до програми.

Забезпечуючи дотримання вимог як до програмного, так і до апаратного забезпечення, можна ефективно впровадити інформаційну систему для автоматизації бізнес-процесів, гарантуючи оптимальну продуктивність, безпеку та покращений досвід роботи з користувачем для малого підприємства.

Діаграма розгортання — це тип структурної діаграми в уніфікованій мові моделювання (UML), яка ілюструє фізичне розгортання артефактів (компонентів програмного забезпечення) на вузлах (фізичному обладнанні) у системі. Ця діаграма забезпечує візуальне представлення того, як програмне забезпечення розподіляється в апаратній інфраструктурі, дозволяючи зацікавленим сторонам зрозуміти взаємозв’язок між програмним забезпеченням і апаратним забезпеченням, яке його підтримує.

Діаграми розгортання дають кілька переваг. Вони пропонують чітку візуалізацію фізичної архітектури системи, полегшуючи командам розуміння того, як розгортаються програмні компоненти та як вони взаємодіють з апаратним забезпеченням. Ця чіткість допомагає визначити потенційні вузькі місця, точки збою або проблеми з продуктивністю, пов’язані із середовищем розгортання.

Крім того, діаграми розгортання допомагають у плануванні й управлінні конфігураціями й оновленнями системи. Склавши архітектуру розгортання, команди можуть переконатися, що всі компоненти правильно встановлені та налаштовані, що сприяє більш плавному процесу розгортання [18].

Таким чином, діаграми розгортання є важливими інструментами для моделювання фізичних аспектів програмної системи. Вони покращують розуміння зв’язків між артефактами програмного забезпечення та апаратним забезпеченням, на якому вони розміщені, підтримують планування та керування розгортанням, а також допомагають забезпечити ефективну та надійну роботу систем у призначених середовищах.

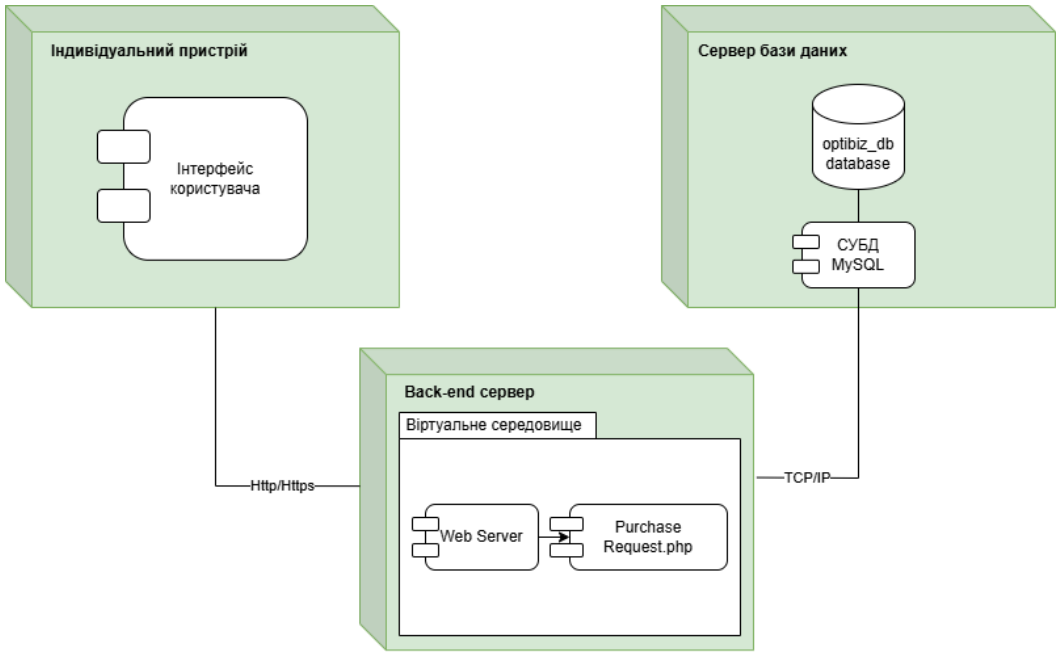

Рис. 3.3 Діаграма розгортання

3.5 Висновки до 3 розділу

У третьому розділі було здійснено комплексне проєктування програмної системи для автоматизації бізнес-процесів малого підприємства. На початковому етапі обґрунтовано вибір системи управління базами даних — MySQL, що забезпечує надійність, швидкодію та зручність інтеграції з PHP та Symfony. Далі визначено архітектуру програмного забезпечення, побудовану за принципом клієнт-серверної моделі з чітким розподілом відповідальностей між фронтендом та бекендом.

Описано організаційну структуру ПЗ, яка включає модулі керування постачальниками, клієнтами, товарами, замовленнями та фінансовими операціями. Особливу увагу приділено вибору інструментарію розробки — PHP,

Symfony, HTML, CSS, JavaScript, PHPStorm — що дозволяє реалізувати ефективну, безпечну та зручну у використанні систему.

Було встановлено вимоги до апаратного та програмного забезпечення для коректної роботи системи, після чого виконано безпосередню реалізацію функціональності. У розділі також описано процес тестування, що дозволив переконатися у правильності виконання основних функцій, стабільності роботи та відповідності системи поставленим вимогам.

У результаті розроблено повноцінне програмне рішення, яке здатне автоматизувати ключові бізнес-процеси підприємства, підвищити ефективність управління ресурсами та покращити взаємодію з клієнтами й постачальниками.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

4.1 Вибір інструментарію для створення програмного забезпечення

Вибір засобів розробки інформаційної системи автоматизації бізнес-процесів малого підприємства здійснювався з урахуванням вимог проєкту,

масштабованості та простоти використання. Вибрані інструменти включають PHP, Symfony, MySQL, Doctrine ORM, HTML, CSS і Bootstrap 5, кожен з яких має унікальні переваги в процесі розробки.

PHP було обрано як основну мову програмування для проєкту через його широке використання у веб-розробці та потужну підтримку сценаріїв на стороні сервера. PHP пропонує багату екосистему бібліотек і фреймворків, що робить його придатним для створення динамічних та інтерактивних веб-додатків. Його легкість у навчанні та гнучкість ще більше підвищують його привабливість для розробки інформаційної системи.

Symfony, надійну структуру PHP, було обрано завдяки її потужним функціям і найкращим практикам, які полегшують розробку масштабованих і підтримуваних програм. Модульна архітектура Symfony дозволяє розробникам створювати повторно використовувані компоненти та сервіси, спрощуючи процес розробки. Крім того, Symfony сприяє використанню шаблону проєктування MVC (Model-View-Controller), який допомагає відокремлювати проблеми всередині програми, створюючи чистіший і організованіший код.

MySQL служить системою керування базою даних для проєкту, надаючи надійне та ефективне рішення для зберігання та пошуку даних. Його природа з відкритим вихідним кодом дозволяє економічно ефективне впровадження, що робить його ідеальним вибором для малих підприємств. Продуктивність MySQL, масштабованість і підтримка складних запитів гарантують, що інформаційна система може обробляти різні навантаження даних у міру зростання бізнесу.

Doctrine ORM було обрано як інструмент об’єктно-реляційного відображення (ORM) для полегшення взаємодії між програмою та базою даних MySQL. Doctrine спрощує операції з базою даних, дозволяючи розробникам працювати з об’єктами PHP замість необроблених запитів SQL, підвищуючи читабельність коду та зручність обслуговування. Його потужні функції, такі як зв’язки сутностей і відкладене завантаження, роблять його цінним доповненням до стеку розробки.

Для розробки інтерфейсу HTML і CSS були обрані як базові технології для створення структури та стилю веб-додатку. HTML забезпечує основну розмітку для вмісту, а CSS покращує візуальне представлення інтерфейсу користувача.

Щоб ще більше покращити взаємодію з користувачем і оптимізувати процес проєктування, Bootstrap 5 було включено як зовнішню структуру. Bootstrap пропонує адаптивну систему дизайну, що дозволяє програмі плавно адаптуватися до різних розмірів екрана та пристроїв. Його готові компоненти та утиліти пришвидшують розробку візуально привабливого та зручного для користувача інтерфейсу, зменшуючи кількість необхідного спеціального CSS.

Таким чином, вибір PHP, Symfony, MySQL, Doctrine ORM, HTML, CSS і Bootstrap 5 як інструментів розробки для інформаційної системи відображає стратегічний підхід до створення надійної, масштабованої та зручної програми. Ці інструменти працюють разом, щоб створити ефективне середовище розробки, яке відповідає потребам малого підприємства, гарантуючи, що кінцевий продукт є функціональним і придатним для обслуговування.

4.2 Розробка системи

Розробка інформаційної системи здійснювалася з використанням сучасних веб-технологій, що забезпечують високу продуктивність, гнучкість та масштабованість. Основою бекенд-частини став мову програмування PHP у поєднанні з фреймворком Symfony, який надає потужні інструменти для побудови складних веб-додатків із чіткою архітектурою та підтримкою шаблонів проєктування.

В процесі розробки був створений набір контролерів для обробки запитів користувачів, реалізовано логіку бізнес-процесів, зокрема управління постачальниками, клієнтами, товарами та замовленнями. Для взаємодії з базою даних MySQL застосовувався компонент Doctrine ORM, що забезпечує зручне і безпечне маніпулювання даними через об’єктно-орієнтований підхід.

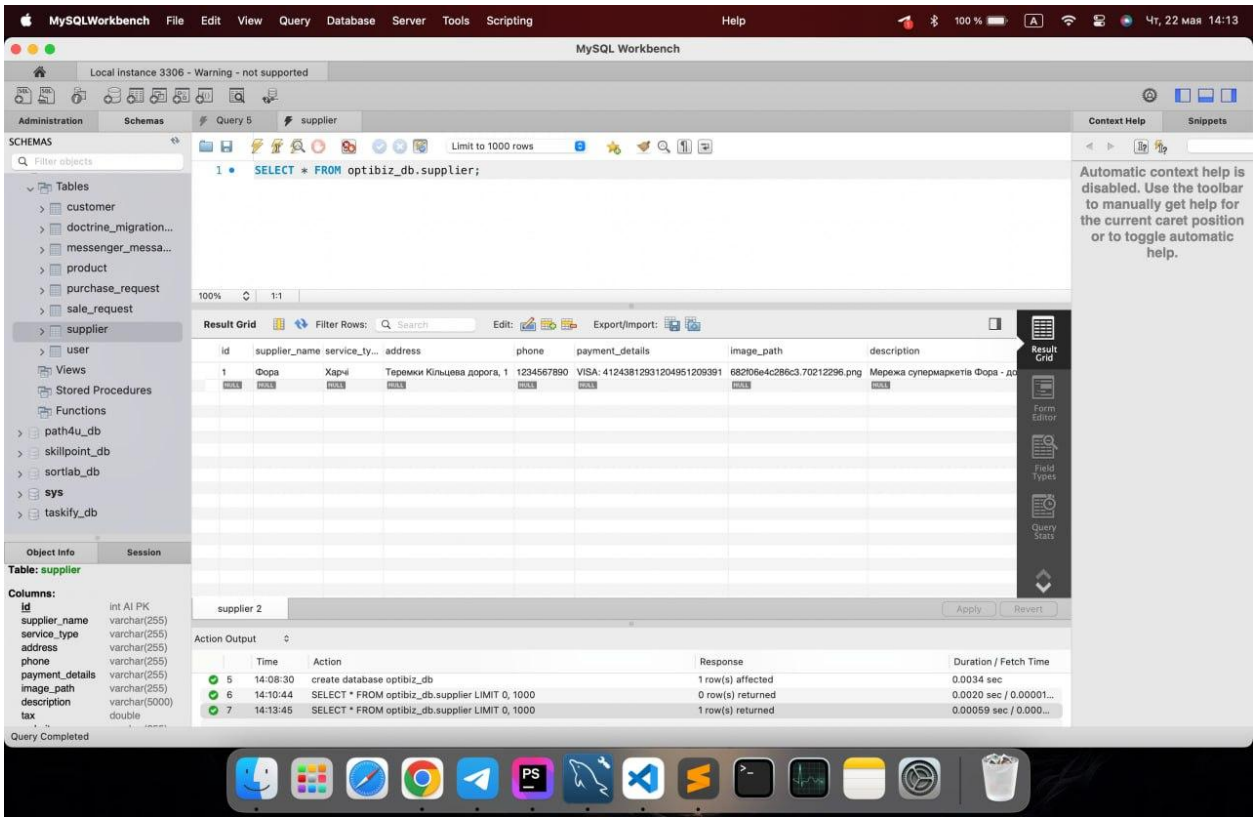

Рис. 4.1 Створення бази даних в СУБД

Інтерфейс користувача реалізовано за допомогою HTML, CSS і JavaScript, що дозволяє створювати інтуїтивно зрозумілий та адаптивний дизайн. Використання сучасних можливостей CSS надало можливість оформити сторінки у відповідності з вимогами зручності та естетики. JavaScript додав інтерактивності, зокрема реалізовано динамічне оновлення даних, валідацію форм та покращення користувацького досвіду.

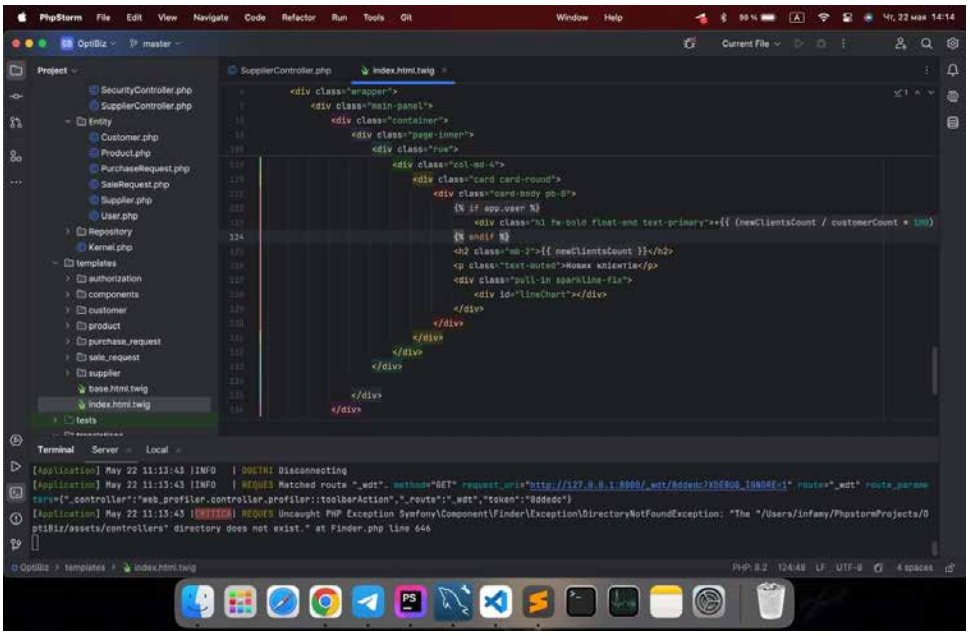

Рис. 4.2 Розробка фронтенду

Для ефективної розробки використовувався середовище програмування PHPStorm, яке значно спростило процес написання коду, налагодження та тестування системи завдяки своїм інструментам автодоповнення, рефакторингу і контролю версій.

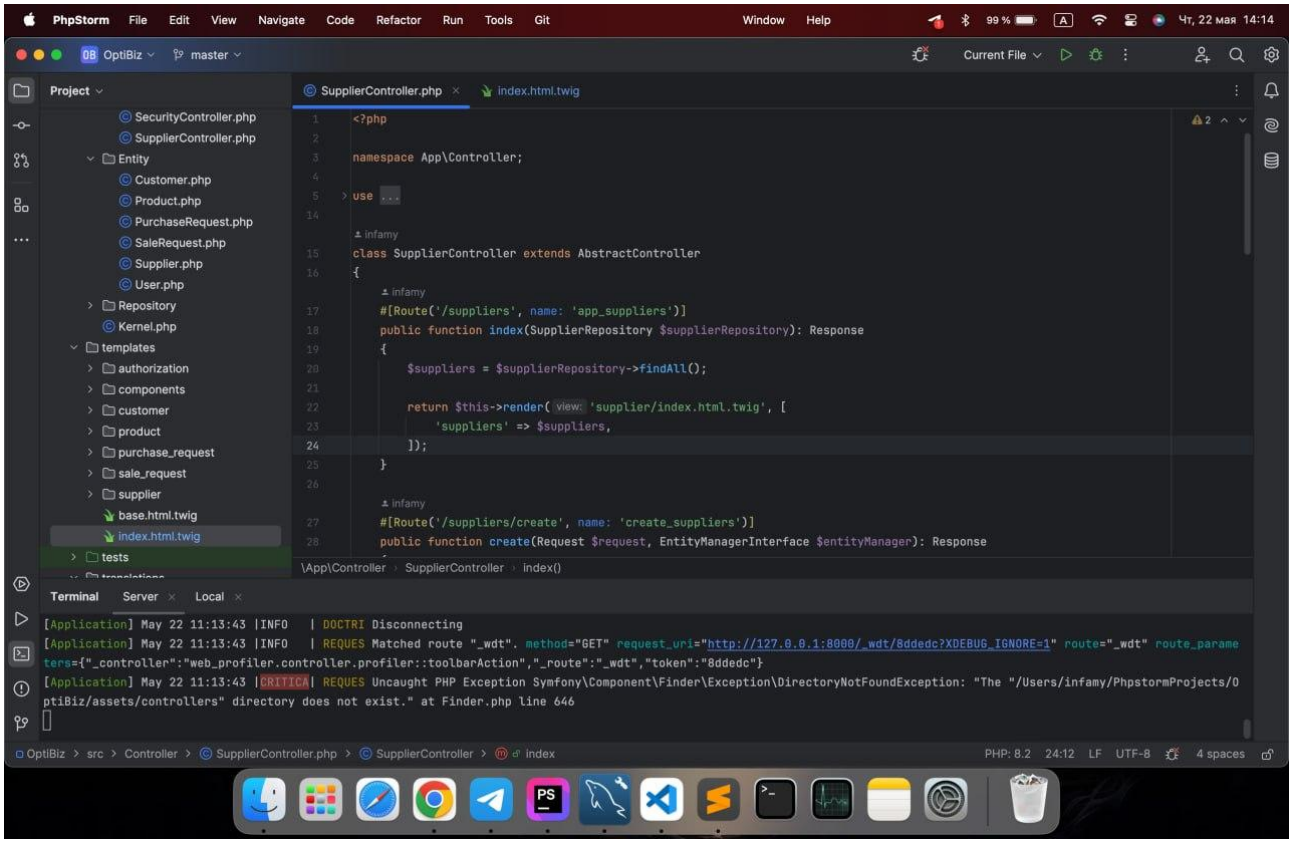

Рис. 4.3 Розробка бекенду

Таким чином, поєднання сучасних технологій і методів розробки дозволило створити надійну, зручну та масштабовану інформаційну систему, яка відповідає потребам малого бізнесу в автоматизації ключових процесів.

4.3 Перевірка якості програмного продукту

Спочатку ми потрапляємо на головний екран системи зображений на рисунку 4.4 На головній сторінці у нас представлені всі елементи керування системою. Тут у нас показуються актуальні значення для постачальників, клієнтів, скільки зароблено, скільки активних замовлень і т.п.

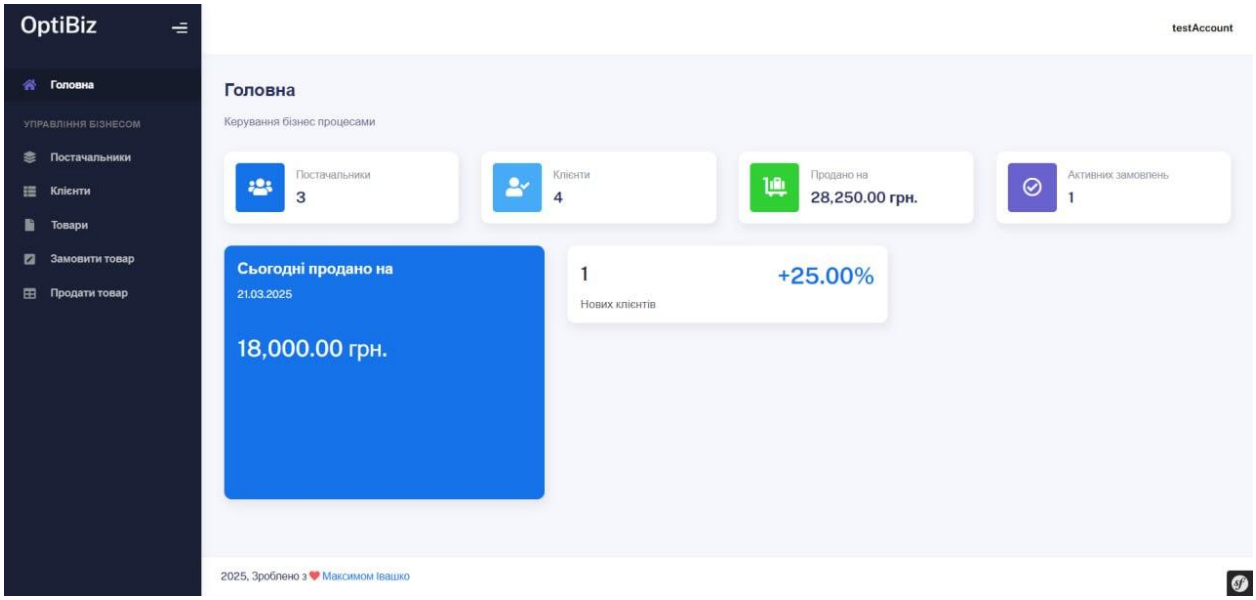

Рис. 4.4 Головна сторінка

На сторінці постачальників ми можемо побачити список доданих постачальників, їх логотип, назву та тип послуг, які вони надають (Рис. 4.5)

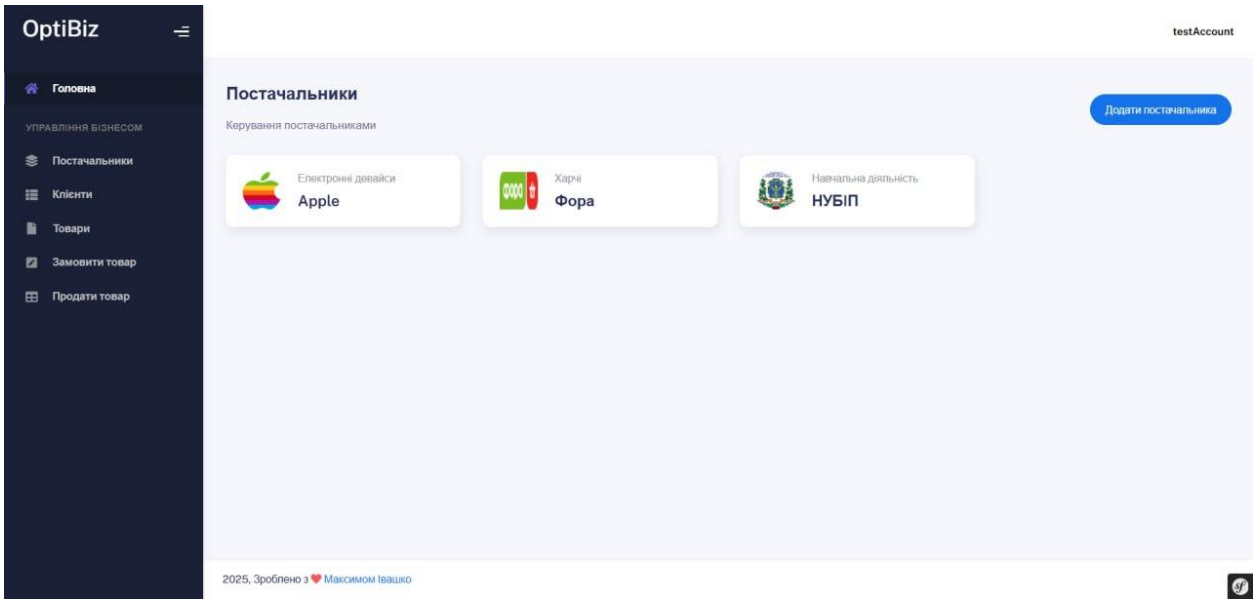

Рис. 4.5 Сторінка “Постачальники”

Натиснувши кнопку "Додати постачальника", ми перейдемо на форму заповнення даних для додавання постачальника. Тут нам треба заповнити всю необхідну інформацію про нього та підтвердити дію (Рис. 4.6).

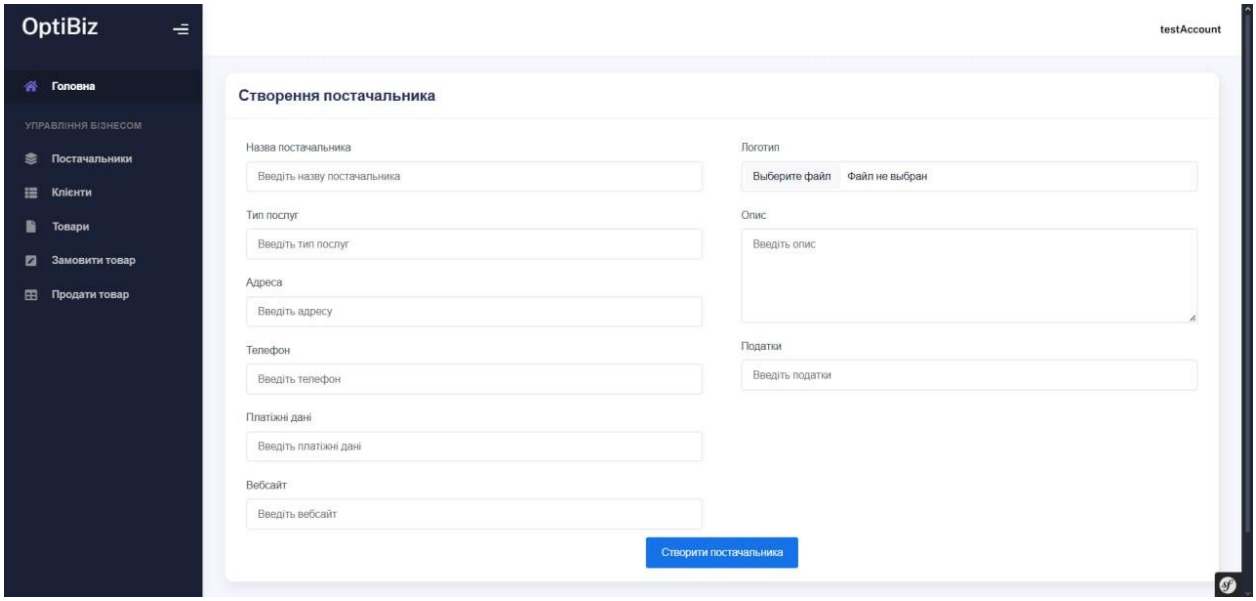

Рис. 4.6 Форма створення постачальника

Успішно створивши постачальника, ми переходимо на його сторінку і можемо переглянути детальну інформацію про нього (Рис. 4.7)

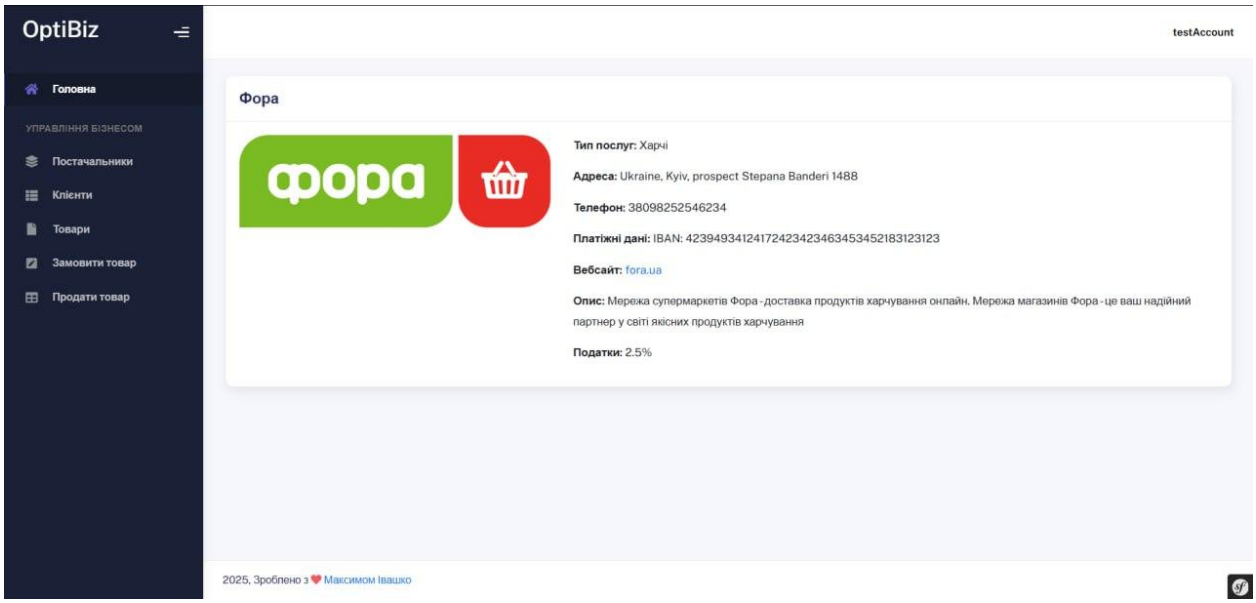

Рис. 4.7 Сторінка створеного постачальника

Тепер переходимо на сторінку клієнтів (Рис. 4.8). Тут ми маємо список із доданих у систему клієнтів, яким ми продаємо свої товари чи послуги. У картці клієнтів показується їхній представник, його ім'я, контактні дані та можливість написати клієнту або видалити його.

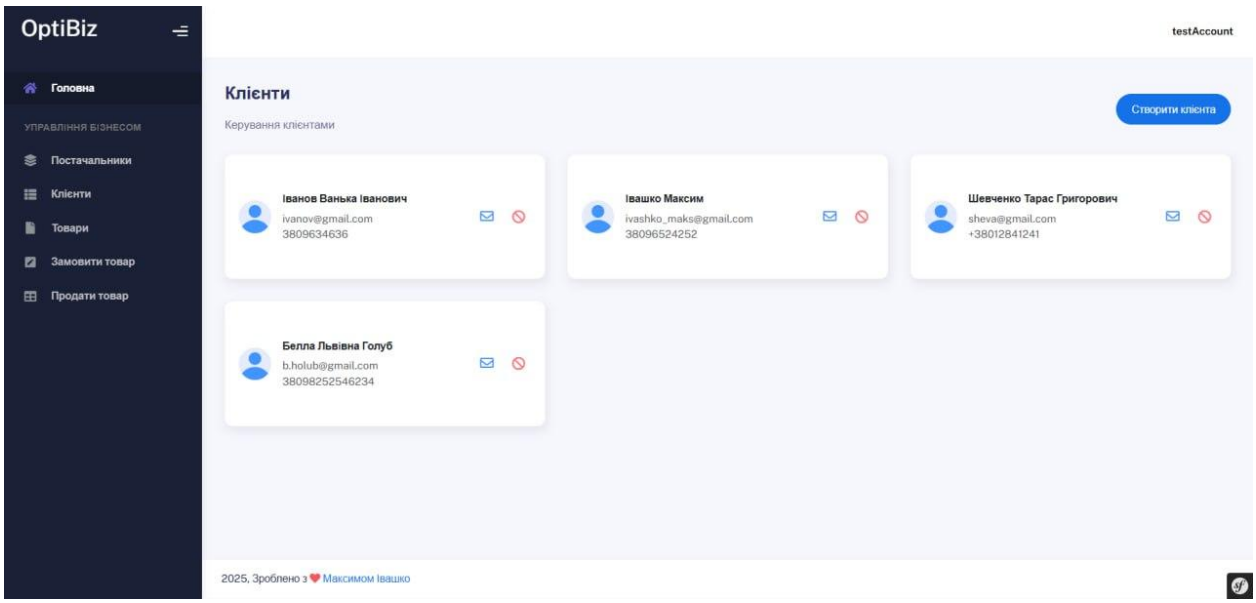

Рис. 4.8 Сторінка “Клієнти”

Якщо ми додамо нового клієнта, ми можемо перейти на його сторінку і переглянути всі дані та інформацію про нього (Рис. 4.9).

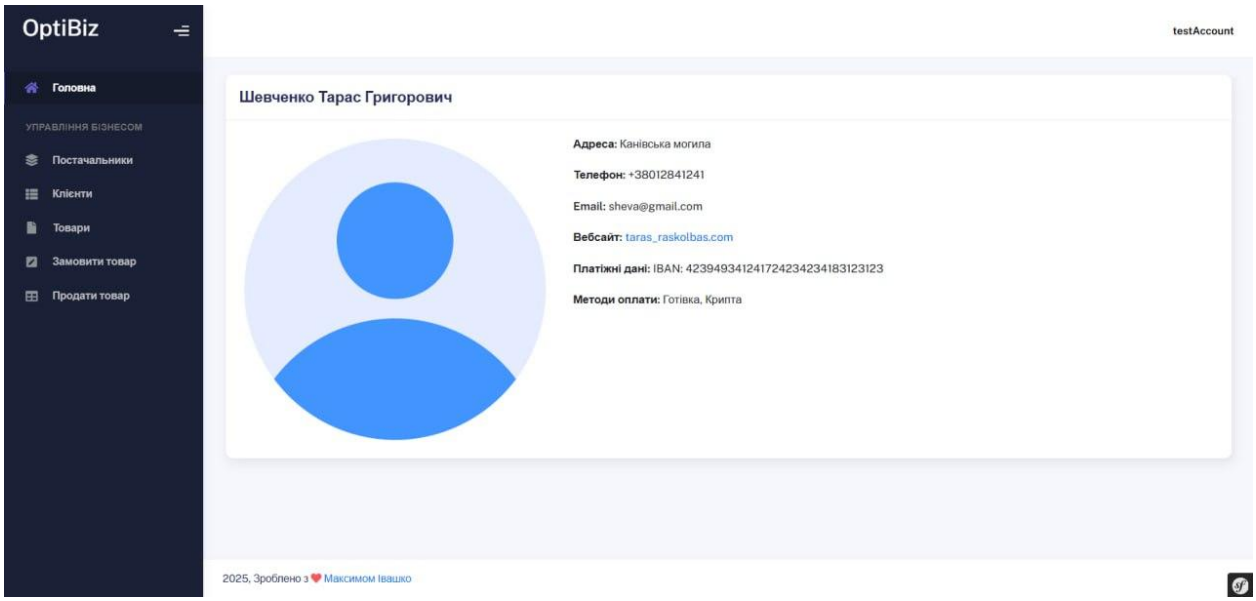

Рис. 4.9 Сторінка створеного клієнта

Переходимо сторінку "Товари" (Рис. 4.10). Тут ми бачимо список створених товарів, з якими ми можемо працювати: створювати заявки на купівлю чи продаж.

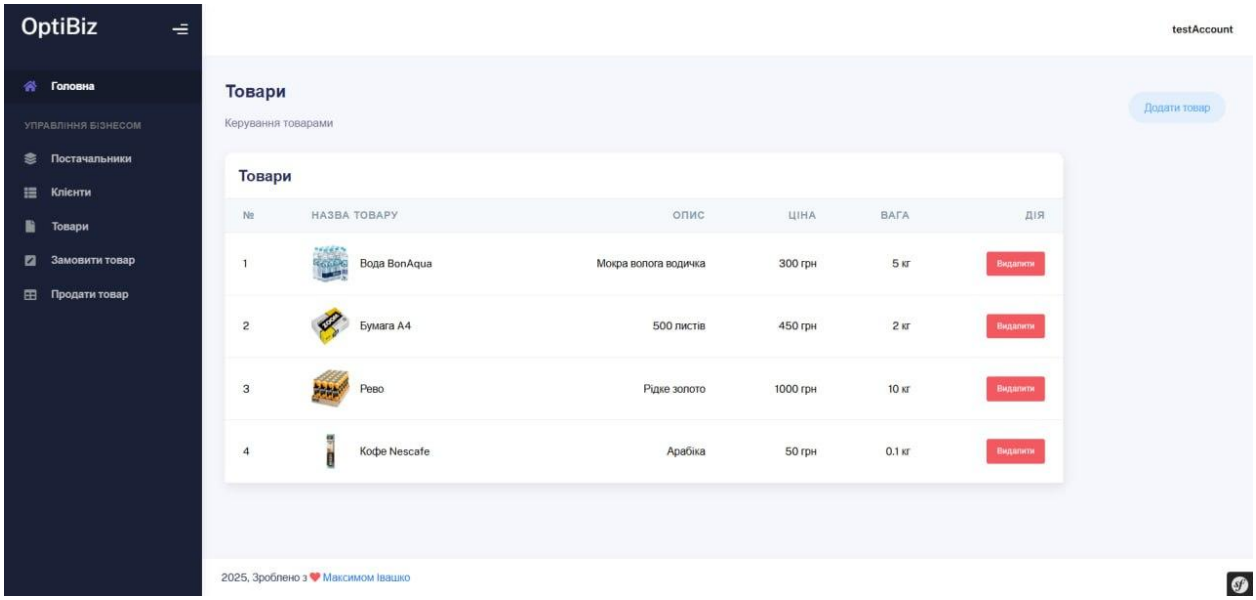

Рис. 4.10 Сторінка “Товари”

Натиснувши кнопку "Додати товар" ми перейдемо на сторінку створення товару, де нам потрібно заповнити всі необхідні поля та успішно підтвердити свою дію (Рис. 4.11).

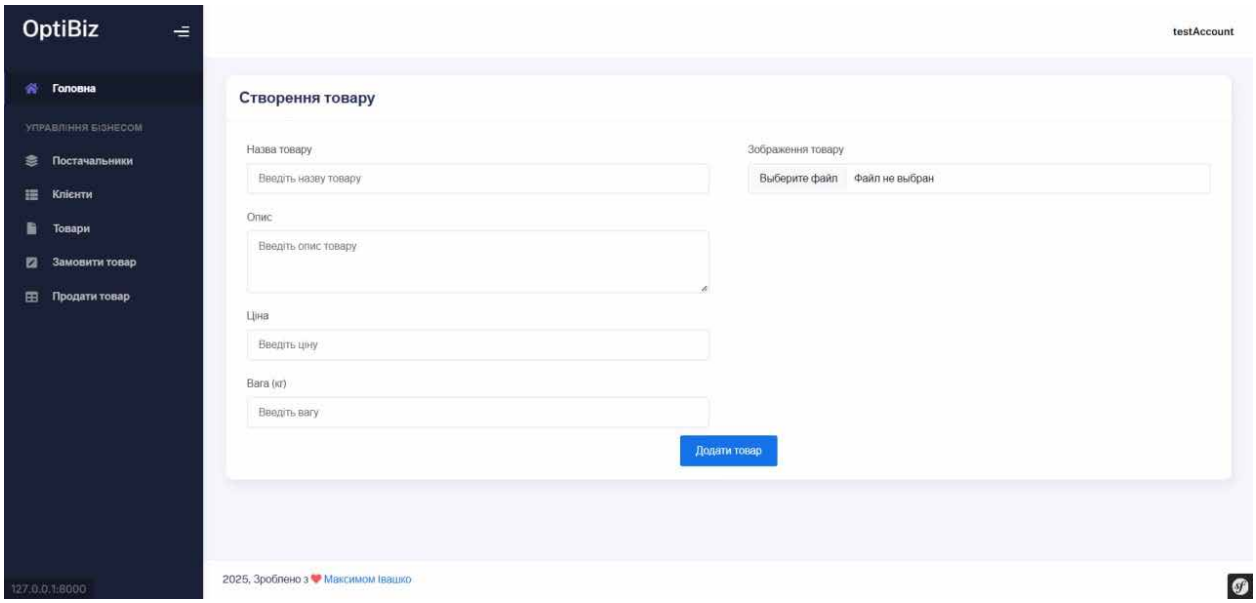

Рис. 4.11 Форма створення товару

Переходимо на сторінку "Замовити товар" (Рис. 4.12). Тут у нас є таблиця з усіма замовленнями, які ми вже оформили у постачальників. Тут показуються якісь товари ми купували, їх кількість, ціну, статус доставки, який можна регулювати натиснувши на зелену кнопку.

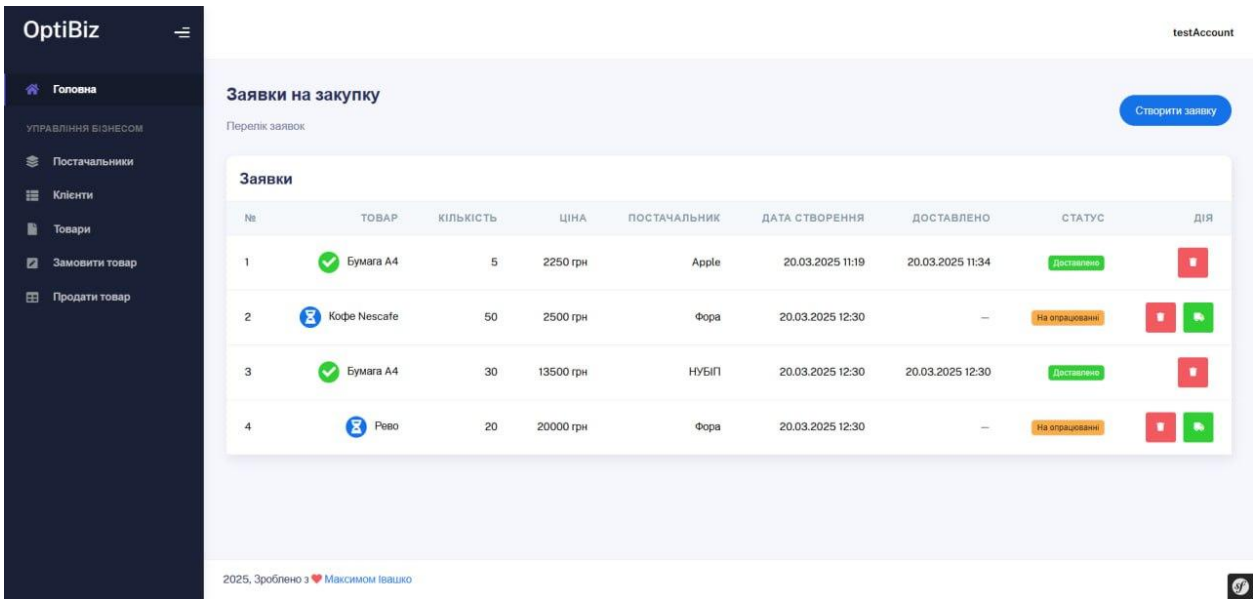

Рис. 4.12 Сторінка “Замовити товар”

Натиснувши "Створити заявку", переходимо на сторінку оформлення замовлення. Тут нам треба вибрати одного з існуючих постачальників та його товари, які нас цікавлять (Рис. 4.13).

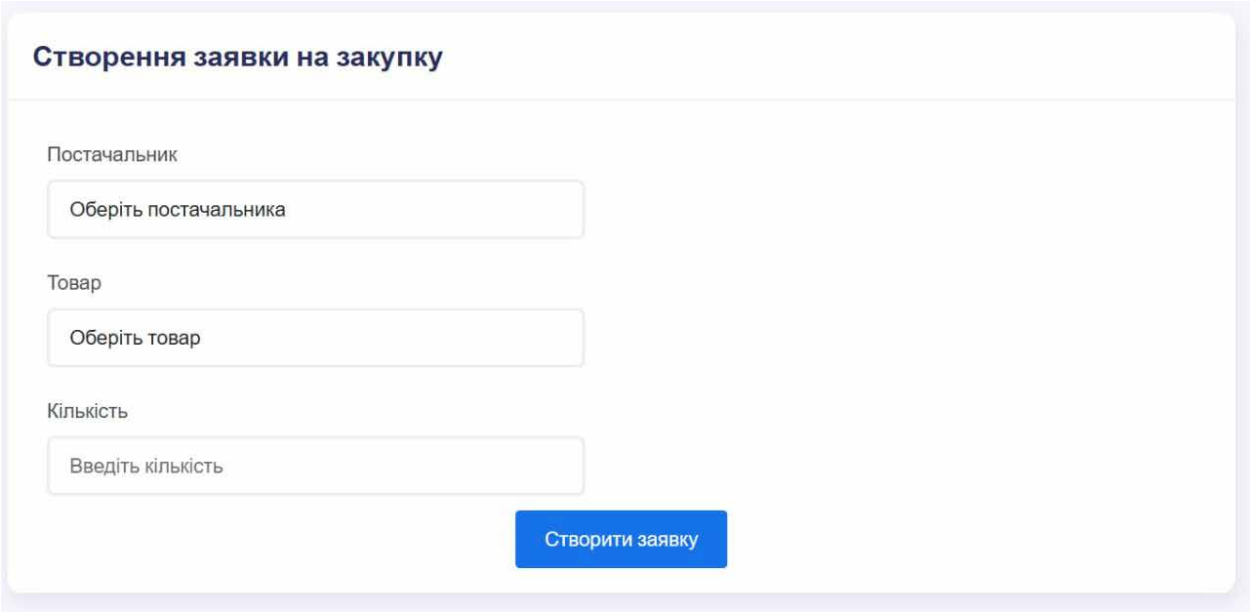

Рис. 4.13 Форма створення заявки на закупку товару

Тепер ми можемо перейти на сторінку продажу товарів клієнту (Рис. 4.14). Тут у нас є таблиця з готовими заявками на продаж, де показується клієнт, статус заявки і всі необхідні дані.

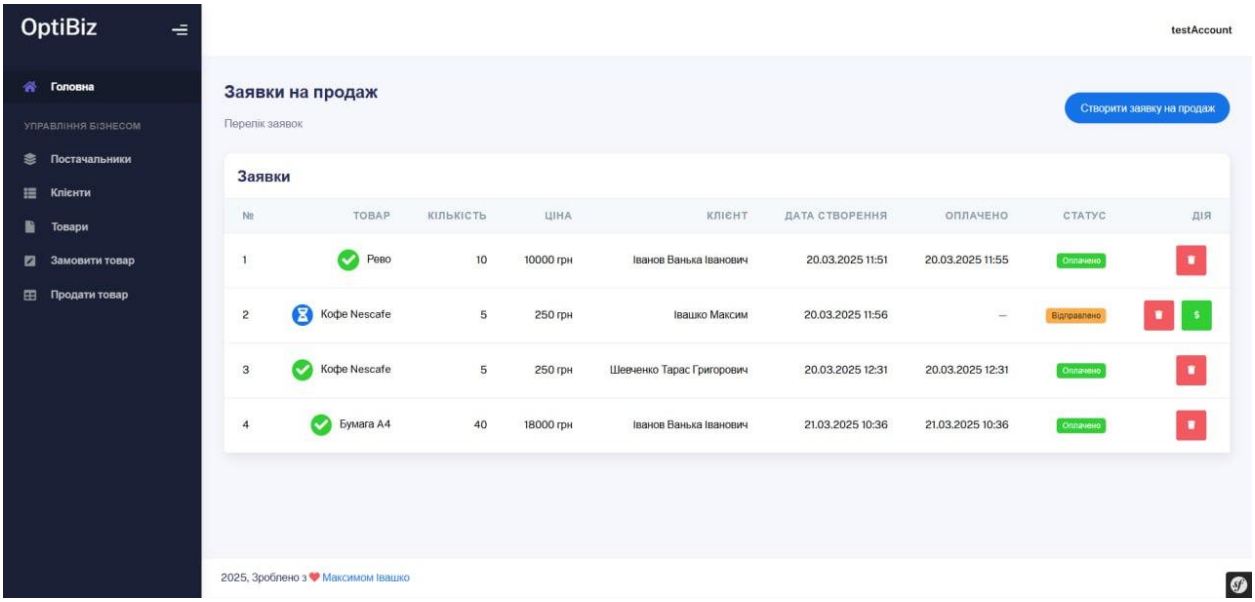

Рис. 4.14 Сторінка “Продати товар”

Також, у системі є можливість реєстрації та авторизації, на Рис. 4.15 у нас представлена форма логіну.

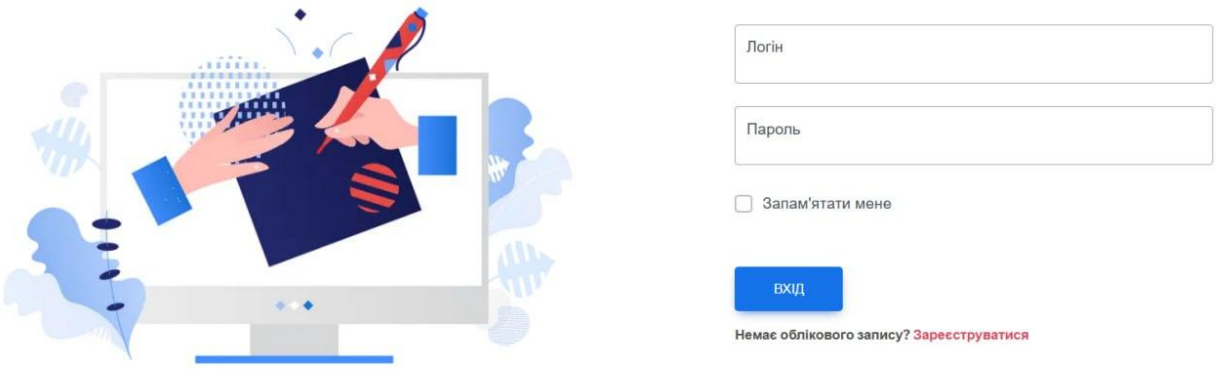

Рис. 4.15 Сторінка авторизації

4.4 Висновки до 4 розділу

У четвертому розділі було розглянуто практичну реалізацію інформаційної системи для автоматизації бізнес-процесів малого підприємства, що є завершальним етапом проєктування та створення програмного забезпечення.

На першому етапі було здійснено обґрунтований вибір інструментарію, що забезпечує ефективну реалізацію поставлених завдань. Основними критеріями вибору стали надійність, популярність серед розробників, активна підтримка спільноти, а також сумісність технологій. Для бекенду обрано мову програмування PHP у зв’язці з фреймворком Symfony, який забезпечує модульність, безпеку та гнучкість розробки. Для роботи з базою даних використано MySQL у поєднанні з ORM-бібліотекою Doctrine, що спростило процес моделювання та доступу до даних. Для розробки клієнтської частини застосовано HTML, CSS, JavaScript, а також фреймворк Bootstrap 5, що дозволив створити адаптивний і зручний інтерфейс.

У підрозділі 4.2 було висвітлено етапи розробки системи, що охоплювали створення моделей сутностей, реалізацію логіки взаємодії між користувачами, постачальниками, клієнтами, товарами та заявками. Особливу увагу було приділено побудові ефективної структури проєкту відповідно до

принципів MVC (Model-View-Controller). Було реалізовано функціонал авторизації та автентифікації користувачів, CRUD-операції для основних сутностей, логіку створення заявок на закупівлю й продаж, а також завантаження та обробку супровідних документів. Розробка велась у середовищі PHPStorm із використанням системи контролю версій Git.

Останнім етапом став процес тестування розробленої системи, описаний у підрозділі 4.3. Було проведено функціональне тестування ключових модулів з метою перевірки коректності їх роботи відповідно до вимог. Ручне тестування дозволило виявити та усунути дрібні помилки, а також перевірити зручність користувацького інтерфейсу. Усі основні компоненти системи показали стабільну роботу, правильну обробку вхідних даних, відсутність критичних помилок та відповідність заданій функціональності. Окрему увагу було приділено тестуванню взаємодії між користувачами та перевірці цілісності даних у базі.

ВИСНОВКИ

У межах дипломного проєкту було реалізовано повний цикл розробки інформаційної системи для автоматизації бізнес-процесів малого підприємства. Для досягнення поставленої мети було використано комплекс сучасних методів, підходів та засобів, що охоплюють як етапи аналізу предметної області, так і безпосереднє проєктування та реалізацію програмного забезпечення.

У процесі розробки було застосовано методи об’єктно-орієнтованого аналізу та моделювання, функціонального моделювання, структурного проєктування та каскадного підходу до життєвого циклу програмного продукту. На етапі аналізу було використано порівняльний аналіз аналогічних рішень, методи постановки задачі та формулювання вимог, що дозволило сформувати чітке бачення майбутньої системи. Для моделювання використовувалися UML-

діаграми, які надали змогу формалізувати структуру та поведінку системи ще до етапу реалізації.

Інструментальне забезпечення проєкту включало мови та технології PHP, Symfony, MySQL, Doctrine ORM, HTML, CSS, Bootstrap 5, JavaScript та середовище розробки PHPStorm. Symfony як потужний фреймворк дозволив дотримуватися архітектурного патерну MVC, що забезпечило логічну структурованість і підтримуваність коду. MySQL виступала в ролі надійного СУБД для зберігання структурованих даних. Завдяки ORM Doctrine було реалізовано ефективну роботу з базою даних без необхідності безпосереднього написання SQL-запитів, що підвищило безпечність і стабільність системи. Для візуального оформлення інтерфейсу застосовувались CSS та Bootstrap 5, що дало змогу створити адаптивний та зручний дизайн.

Під час реалізації проєкту були виконані наступні завдання, поставлені у вступі:

- проведено аналіз предметної області: визначено основні бізнес-процеси, які потребують автоматизації; виявлено недоліки ручного ведення операцій на малому підприємстві; проаналізовано існуючі інформаційні системи, виявлено їх обмеження;
- побудовано функціональну та об'єктну модель системи: створено діаграми варіантів використання, діаграми послідовностей, діаграми класів, що дозволило візуалізувати ключові процеси системи та визначити взаємозв'язки між сутностями;
- розроблено архітектуру та логічну модель програмної системи: проєкт включав проєктування структури бази даних, опис компонентів, зв’язків між ними, а також розподіл обов’язків у коді;
- здійснено безпосередню розробку інформаційної системи: реалізовано функціональність додавання, редагування та перегляду постачальників, товарів, клієнтів, а також оформлення заявок на закупівлю й продаж товарів;

- виконано тестування розробленої системи: перевірено коректність функцій, проведено валідацію вхідних даних, протестовано взаємодію з базою даних, перевірено працездатність інтерфейсу.

У результаті проєктування вдалося досягти високого рівня структурованості, гнучкості та масштабованості розробленого програмного продукту. Розроблена система автоматизує критично важливі бізнес-процеси, зокрема облік товарів, керування замовленнями, взаємодію з постачальниками та клієнтами, що суттєво скорочує витрати часу та зменшує ймовірність помилок.

Узагальнюючи, можна зробити висновок, що інформаційна система, розроблена в межах цього дипломного проєкту, відповідає поставленим вимогам та меті дослідження. Вона є не просто технічним рішенням, а й ефективним інструментом управління бізнесом. Запропоноване рішення може бути успішно впроваджене на підприємствах малого та середнього бізнесу для підвищення продуктивності праці, автоматизації документообігу та ефективного прийняття рішень.

Майбутній розвиток системи може включати інтеграцію з платіжними системами, реалізацію аналітичного модуля для оцінки ефективності бізнесу, мобільну версію інтерфейсу та впровадження багатомовності для розширення ринку. Таким чином, створена система має не лише прикладне, а й стратегічне значення для підприємства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Програма для управління витратами організації SAP Ariba – [Електронний ресурс] – Режим доступу: https://www.sap.com/cis/products/spend-management.html
2. Precoro – [Електронний ресурс] – Режим доступу: https://precoro.com
3. Best-in-class Inventory management features TradeGeeko – [Електронний ресурс] – Режим доступу: https://www.tradegecko.com/product-tour/inventory-management-system
4. Діаграма варіантів використання (UseCase diagram) – [Електронний ресурс] – Режим доступу: https://flexberry.github.io/ru/fd_use-case-diagram.html
5. Застосування UML (частина 2). Діаграма послідовності – [Електронний ресурс] – Режим доступу: https://dut.edu.ua/ua/news-1-626-7897-zastosuvannya-uml-chastina-2-diagrama-poslidovnosti---sequence-diagram_kafedra-kompyuternih-nauk-ta-informaciynih-tehnologiy
6. Діаграма активностей (видів діяльності) – [Електронний ресурс] – Режим доступу: https://flexberry.github.io/ru/fd_activity-diagram.html
7. UML-діаграми класів – [Електронний ресурс] – Режим доступу: https://prog-cpp.ru/uml-classes/
8. ER-діаграма (ERD): визначення та огляд – [Електронний ресурс] – Режим доступу: https://www.lucidchart.com/pages/ru/erd-диаграмма
9. How to Connect to a MySQL Database – [Електронний ресурс] – Режим доступу: https://www.domain.com/help/article/how-to-connect-to-a-mysql-

database?psafe_param=1&utm_source=google&utm_medium=genericsearch&gad=1&gclid=Cj0KCQjw98ujBhCgARIsAD7QeAjoPHqpoNQAvN9h0FPMhQ2ru17pLdKmlTedtxpsI2VUl2-BmJZvh50aAl_LEALw_wcB&gclsrc=aw.ds

10. Що таке MySQL? – [Електронний ресурс] – Режим доступу: https://hyperhost.ua/ru/wiki/chto-takoe-mysql?gclid=Cj0KCQjw98ujBhCgARIsAD7QeAioUWB6Q4qgXASSyKP5vL6Z3zsI_8R_OQbpsi-daTiVuCZ8hffJ5aAaAg8vEALw_wcB
11. Вибір мови та середовища програмування. – [Електронний ресурс] – Режим доступу: https://studopedia.org/5-67082.html
12. Багаторівнева архітектура – [Електронний ресурс] – Режим доступу: https://simpleone.ru/glossary/mnogourovnevaya-arhitektura/
13. Діаграма розгортання (Deployment Diagram) – [Електронний ресурс] – Режим доступу: https://ua.kursoviks.com.ua/metodychni_vkazivky/article_post/1648-laboratorna-robota-no8-na-temu-diagrama-rozgortannya-deployment-diagram
14. CA Technologies. (n.d.). Erwin Data Modeler. [Електронний ресурс] – Режим доступу: https://www.ca.com/us/products/erwin-data-modeler.html
15. Rumbaugh J., Jacobson I., Booch G. Unified Modeling Language Reference Manual. Pearson Education, Limited, 2004. 752 p.
16. Microsoft. (2021). Model-View-ViewModel (MVVM) architectural pattern. [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/windows/uwp/data-binding/data-binding-and-mvvm
17. Microsoft Docs. (2021). Layered architecture pattern – [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/patterns/layered
18. What is Component Diagram? – [Електронний ресурс] – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/
19. Rumbaugh J., Jacobson I., Booch G. Unified Modeling Language Reference Manual. Pearson Education, Limited, 2004. 752 стор.

20. Соммервіль І. Розробка програмного забезпечення. 10-е вид. Addison-Wesley, 2015. 816 с.
21. Прессман Р. С. Розробка програмного забезпечення: підхід спеціаліста. 9-е вид. McGraw-Hill, 2010. 960 с.
22. Ambler S. W. The Object Primer: Agile Model-Driven Development. 3-е вид. Cambridge University Press, 2012. 352 с.
23. Фаулер М. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3-е вид. Addison-Wesley, 2004. 272 с.
24. Blanchard B. S., Fabrycky W. J. Системна інженерія та аналіз. 4-е вид. Prentice Hall, 2010. 800 с.
25. Мердок Дж. А., Тернер А. М. Автоматизація бізнес-процесів: Посібник з автоматизації вашого бізнесу своїми руками. Entrepreneur Press, 2009. 208 с.
26. Ambler S. W. Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. Wiley, 2002. 288 с.
27. Kuo R. J., Yang C. Y. Огляд управління бізнес-процесами: концепції, технології та застосування. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 2009.
28. Чаффі Д. Цифровий бізнес і управління електронною комерцією. 6-е вид. Pearson Education, 2015. 624 с.
29. Laudon K. C., Laudon J. P. Системи управління інформацією: управління цифровою фірмою. 15-е вид. Pearson, 2018. 480 с.
30. Хоффер Дж. А., Джордж Дж. Ф., Валачіч Дж. С. Сучасний системний аналіз і проєктування. 8-е вид. Pearson, 2019. 864 с.
31. Кім Х. Дж., Лі Дж. Х. Роль інформаційних технологій в управлінні бізнес-процесами. Міжнародний журнал інформаційних систем та управління проєктами, 2017.
32. Хаммер М., Чемпі Дж. Реінжиніринг корпорації: маніфест революції в бізнесі. Harper Business, 1993. 240 с.