НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук

_____/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення інформаційної облікової системи постачання будівельних матеріалів»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
_____к.т.н., доцент_____ / Голуб Б.Л./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи :
д.т.н.,проф._____ / Хиленко В.В./
підпис ПІБ

Виконав:_____ / Маньковський Д.А./
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
підпис

" 16 " грудня 2024 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студента Маньковського Данила Андрійовича

Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення інформаційної облікової системи постачання будівельних матеріалії

Затверджена наказом ректора НУБіП України № 2248 "С " від 16 грудня 2024р

2. Термін подання завершеної роботи на кафедру 2025 . 05 . 25
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

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

Керівник бакалаврської кваліфікаційної роботи_____ / Хиленко В.В. /
підпис ініціали та прізвище

Завдання прийняла до виконання _____ / Маньковський Д.А. /
підпис ініціали та прізвище

Дата отримання завдання 2024 . 12 . 16
рік, місяць, число

ЗМІСТ

ВСТУП

Необхідність покращення процесів виробництва та торгівлі змушує сучасні підприємства звертатись до ринку інформаційних технологій, зокрема до розробників систем управління інформацією. Основні причини цього такі:

По-перше, автоматизація бізнес-операцій дозволяє ефективно та комплексно керувати підприємством. Автоматизація значно знижує кількість помилок, що допускаються працівниками під час роботи. Таким чином, автоматизація практично усуває людський фактор помилок.

По-друге, автоматизація бізнесу значно зменшує ймовірність збитків, мінімізує ризики та економить трудові ресурси. Вона усуває проблеми, пов’язані з перевантаженням, економить час на ручному обліку та спрощує обробку документів.

Метою цієї дипломної роботи є розробка програмного забезпечення для системи інформаційного обліку постачання будівельних матеріалів.
Для досягнення цієї мети необхідно виконати такі завдання: проаналізувати предметну область та створити моделі бізнес-процесів із використанням UML-діаграм; проаналізувати інструменти розробки і вибрати необхідний набір для побудови системи управління інформацією магазину; спроектувати базу даних із використанням сервера MS SQL; реалізувати інтерфейс користувача для магазину будівельних матеріалів на обраній платформі; провести аналіз системи.

Обов’язковим елементом системи управління інформацією та автоматизації підприємства є база даних. Моделювання даних — один із ключових етапів процесу автоматизації бізнесу.

У магазинах, що спеціалізуються на продажу будівельних матеріалів, завжди була і досі є потреба у простому, безкоштовному та зручному програмному рішенні для обліку матеріалів та контролю їх продажів.

Впровадження системи управління інформацією дозволяє вирішити кілька важливих проблем: скоротити час обробки замовлень, зменшити час на

перевірку наявності та кількості будівельних матеріалів, спростити роботу з базою даних клієнтів, підвищити якість обробки даних та знизити кількість помилок працівників. Тому поставлено мету створити безкоштовну, просту у використанні систему, яка зробить роботу персоналу магазину комфортнішою та ефективнішою.

Під час розробки роботи створено систему управління інформацією для реєстрації, обліку та обробки замовлень будівельних матеріалів. Система була розроблена відповідно до технічних вимог проєкту. Спроектовано просту та зручну базу даних.

Розроблена система управління інформацією дозволяє персоналу використовувати ПЗ у повсякденній роботі, автоматизуючи процес обробки замовлень і обліку продажів, а також полегшуючи створення бази даних клієнтів для покращення обслуговування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задачі

Ця система призначена для підтримки магазинів, які займаються продажем будівельних матеріалів.
Основні цілі програмного забезпечення:

- спростити роботу персоналу;
- забезпечити швидку та зручну перевірку наявності товарів;
- вести облік товарів у магазині.

Наразі процес обробки замовлень — включно з їх прийомом, реєстрацією та опрацюванням — пов’язаний з деякими незручностями, такими як:

- велика кількість повторюваних операцій, які виконують працівники;
- відсутність зручного та точного методу контролю за запасами.

Хоча ця інформаційна система не забезпечує повної автоматизації всього процесу обліку та продажів, вона пропонує користувачам простий і зручний інтерфейс разом із засобами для виконання їхньої роботи.

Функції, які має виконувати розроблена система управління інформацією:

- робота з каталогом товарів;
- зберігання інформації про наявність товарів;
- зберігання інформації про замовлення та клієнтів;
- генерація квитанцій і рахунків;
- створення звітів.

1.2 Моделювання предметної області

Моделювання предметної області — це перший і дуже важливий етап у

проєктуванні програмної системи для будь-якого підприємства або магазину. Сьогодні ринок програмного забезпечення пропонує широкий спектр CASE-інструментів для цих цілей. Значна частина (до 80%) загальних витрат на впровадження системи припадає на дизайн і моделювання. Тому правильно побудовані і відповідні моделі (включно з моделями предметної області) відіграють фундаментальну і навіть вирішальну роль у процесі розробки та впровадження ПЗ.

Виправити помилки на етапі моделювання набагато простіше, ніж на наступних етапах розробки. Невідповідна, дисбалансна або неповна модель може призвести до провалу всього проєкту. Щоб уникнути цього та спростити і покращити процес моделювання, за останні роки було розроблено різні сучасні мови моделювання.

Для моделювання системи інформації для обліку продажів будівельних матеріалів використано Уніфіковану мову моделювання (UML). UML було спеціально створено для вирішення таких завдань. Вона базується на методах аналізу та проєктування, орієнтованих на об’єкти (OOA&D), які з’явилися наприкінці 80-х — на початку 90-х років, і стала глобальним стандартом, якого дотримуються більшість розробників систем та прикладного ПЗ.

UML використовують не лише аналітики та дизайнери систем, а й програмісти та тестувальники ПЗ. Ринок інструментів, що базуються на UML і автоматизують процеси розробки ПЗ, постійно розширюється. Серед найвідоміших інструментів — MagicDraw UML, Objecteering, Visual UML та інші.

UML — це візуальна мова моделювання загального призначення, призначена для специфікації, візуалізації, проєктування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. UML є відносно простою, але потужною моделлю, що успішно використовується для побудови концептуальних, логічних і графічних моделей складних систем для різних цілей. Вибір інструментів для моделювання складних систем повністю залежить від конкретних завдань, які ці моделі повинні розв’язувати.

Основні принципи побудови моделей складних систем:

- абстракція: модель має включати лише ті аспекти системи, які безпосередньо стосуються її функціональності. Всі другорядні аспекти опускаються, щоб уникнути ускладнення аналізу та дослідження моделі.
- мультимоделювання: жодна окрема модель не може адекватно описати всі стани, функції та поведінку складної системи. Навіть дуже деталізована модель відображає лише окремі аспекти поведінки або структури системи. Тому для повного подання різних характеристик складної системи завжди потрібні різні моделі.

UML є ключовою частиною сучасного процесу розробки програмного забезпечення. Це універсальна відкрита мова стандарту, що використовує графічні символи для побудови абстрактної моделі системи, відомої як UML-модель.

Як уже згадувалося, UML — це не мова програмування, а мова моделювання. Це інструмент для створення та аналізу UML-моделей. Ці моделі подаються у вигляді спеціальних графічних конструкцій — діаграм.

UML 1.5 визначає дванадцять типів діаграм, які групуються у три категорії:

- чотири типи діаграм представляють статичну структуру додатку;
- п’ять типів діаграм — аспекти поведінки системи;
- три типи діаграм — фізичні аспекти роботи системи.

Точна кількість канонічних діаграм не є критичною, оскільки багато застосунків не потребують усіх видів. Ми розглянемо лише ті діаграми, які використовуватимуться в цьому проєкті.

1.3 Діаграма прецендентів

Одна з UML-діаграм, яка використовується на початковому етапі

проєктування логічної моделі інформаційної системи — діаграма випадків використання.

Айвар Якобсон, один із творців UML, підкреслив важливість випадків використання настільки, що вони стали центральним елементом у розробці систем і плануванні проєктів. Випадок використання — це послідовність дій (транзакцій), які система виконує у відповідь на подію, ініційовану зовнішнім суб’єктом (актором). Випадок використання описує типовий сценарій взаємодії користувача із системою.

Ця діаграма призначена для концептуального моделювання того, як система функціонує у своєму середовищі. Вона складається з кількох акторів, випадків використання, які обмежені межами системи (зазвичай зображено прямокутником), асоціацій між акторами та випадками використання, а також зв’язків між випадками використання і узагальнень між акторами.

Простіше кажучи, діаграма випадків використання служить для опису сервісів, які система надає актору. Кожен випадок використання визначає дії системи під час взаємодії з актором, не деталізуючи, як саме ця взаємодія реалізується.

Основні елементи діаграми випадків використання:

- Межа системи — прямокутник, що охоплює випадки використання, визначаючи межі моделюваної системи (називається суб’єктом або контекстом у UML 2);
- Актор — елемент, що представляє ролі користувачів, які взаємодіють із компонентами системи;
- Випадок використання — графічний елемент (овал із назвою), що представляє окремий випадок використання. Він визначає послідовність дій, які виконує система для надання корисного результату актору;
- Асоціація — лінія, що з’єднує актора з випадком використання, вказуючи, що актор взаємодіє з системою через цей випадок використання;

- Відношення Include — вказує, що базовий випадок використання включає поведінку іншого випадку використання в певній точці свого виконання. Використовується, коли кілька випадків використання мають спільні кроки;
- Відношення Extend — вказує, що поведінка одного випадку використання може розширюватися поведінкою іншого випадку використання за певних умов;
- Узагальнення — показує, що один актор є спеціалізацією іншого і успадковує його взаємодії з випадками використання.

Діаграма випадків використання використовується для опису зовнішньої функціональності системи, розуміється як взаємодія між системою та зовнішнім світом. Вона визначає, що система повинна робити, не входячи в подробиці реалізації функціоналу.

В системі, яка буде розроблятися, існують п’ять основних акторів:

- Клієнт;
- Менеджер;
- Бухгалтер;
- Комірник;
- Логіст.

Клієнт переглядає та вибирає будівельні матеріали та робить замовлення.

Менеджер редагує всі дані по товару, додає нові та видаляє застарілі або невірні дані, замовляє та отримує товар у постачальників.

Бухгалтер веде облік фінансових операцій.

Комірник оформлює накладні на видачу або надходження товарів.в

Логіст планує та координує доставку будівельних матеріалів.

Для предметної області даної інформаційної системи можна виділити наступних акторів, які описані в таблиці 1. та рис. 1.1

Опис акторів

Таблиця 1

Актор	**Короткий опис**
Клієнт	Особа, яка здійснює покупку або замовлення.
Менеджер	Особа, яка відповідає за наявність та стан товару в магазині.
Бухгалтер	Особа, яка відповідає за формування фінансової звітності
Комірник	Особа, яка відповідає за інвентаризацію складу
Логіст	Особа, яка відповідає за доставку матеріалів до клієнта

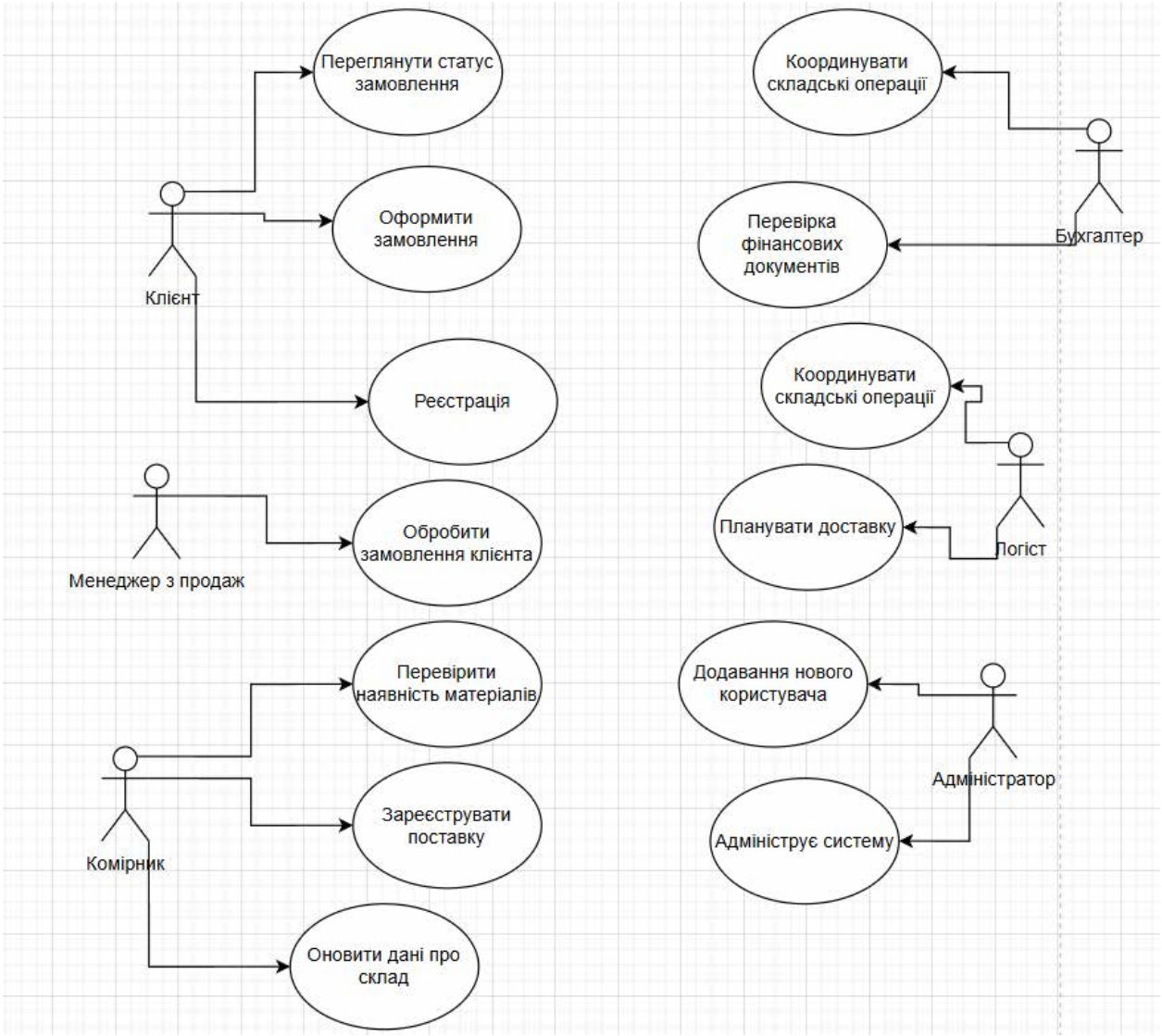


Рис.1.1 Діаграма прецендентів

1.4 Діаграма активностей

Діаграма активностей — це свого роду аналог блок-схеми, де замість процедур системи відображаються конкретні дії.

Ці діаграми широко використовуються для опису поведінки, що включає велику кількість паралельних процесів. Як і діаграми станів та переходів, вони зображуються у вигляді орієнтованого графа, де вузлами є дії, а ребрами — переходи між діями.

Кожна дія в активності може виконуватися один, два або декілька разів під час одного виконання активності. Дії отримують дані, перевіряють їх, трансформують, і деякі дії потребують певної послідовності виконання.

Використання діаграми активностей — зручний спосіб представлення бізнес-процесів у вигляді алгоритмів, які керують операціями промислового підприємства, бізнесу або магазину. Ці алгоритми становлять основу, на якій має будуватися інформаційна система, автоматизуючи певні її частини. Перед розробкою системи необхідно визначити алгоритми робочого процесу цієї нової послуги.

Програміст і розробник систем має чітко розуміти всі бізнес-процеси підприємства, на які вплине нова система. Інакше можна пропустити важливі деталі, що завадить системі повністю виконати свої функції. Впровадження та інтеграція нової системи можуть бути досить складними. Про це потрібно повідомити клієнту якомога раніше, навіть до початку розробки. Інакше проєкт може виявитися не зовсім успішним: клієнт може зазнати збитків і не отримати необхідних для бізнесу послуг.

Основним елементом у цьому типі діаграм є так звана активність(activity), тобто активний стан системи, під час якого виконується конкретна дія. Після завершення цієї дії відбувається перехід до іншої активності. Також можливі більш складні переходи між активностями, наприклад, перехід, зумовлений подіями. Діаграма має включати символи початку(start) та завершення активності(finish).

Діаграма також може включати паралельний розгалужувач((fork)), який ініціює кілька гілок, що працюють одночасно. Ці гілки пізніше можуть об’єднуватися за допомогою структури, званої об’єднанням (join). Діаграма може також

використовувати вузли прийняття рішень і комбінування для логічного розгалуження і об’єднання(decision).

Специфікація активності може передбачати виконання кількох паралельних логічних потоків і наявність механізмів синхронізації, щоб забезпечити правильну послідовність дій.

Діаграма активностей (рис. 1.2) показує алгоритм формування замовлення матеріалів для будівництва. Схема ілюструє два стани: відсутність товарів або наявність товарів. Якщо товари є в наявності, створюється замовлення. Якщо товари відсутні, обираються інші товари і знову перевіряється їх наявність.

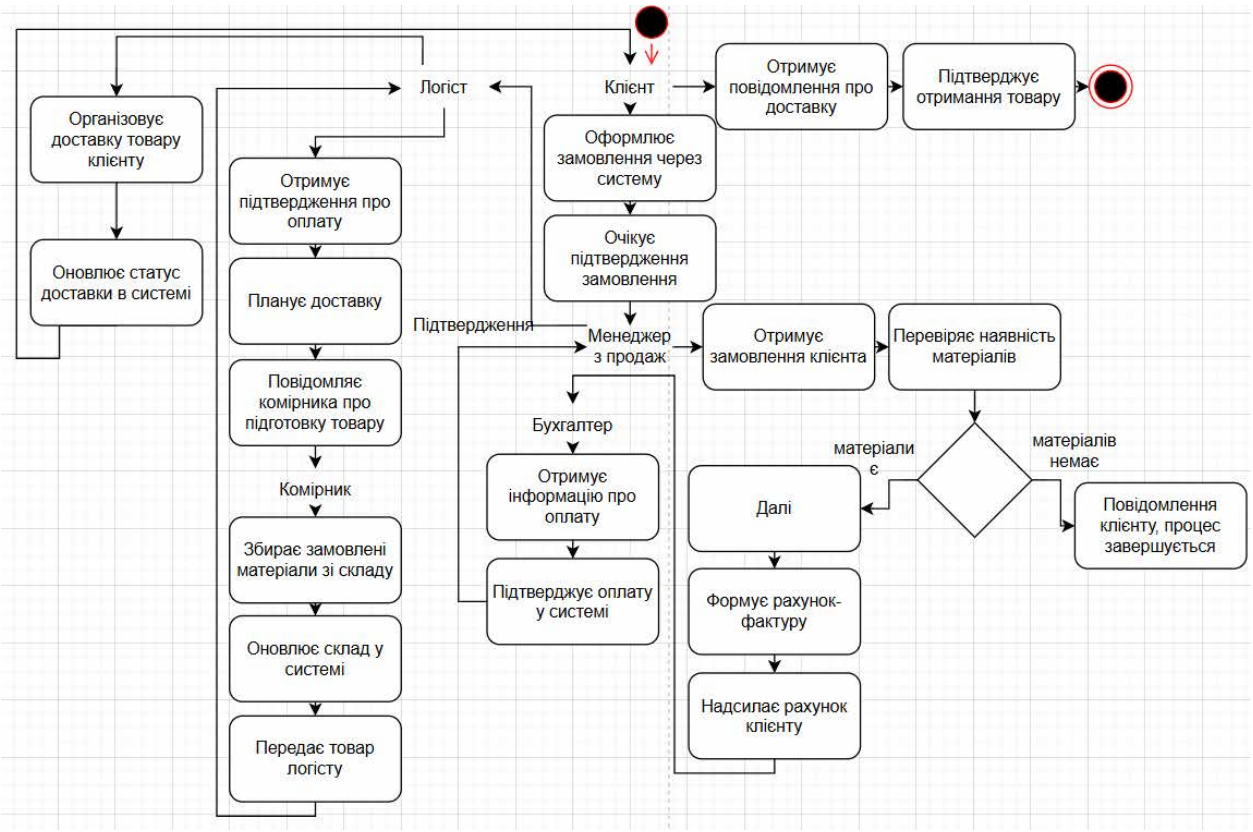


Рис.1.2 Діаграма активностей

1.5 Діаграма послідовності

Діаграма послідовності**(sequence diagram)** — це тип діаграми, що графічно демонструє порядок взаємодій між певними об’єктами програми протягомчасу.

Зазвичай ця діаграма показує, як користувачі (актори з діаграми варіантів використання) взаємодіють з іншими компонентами застосунку під час виконання різних сценаріїв варіантів використання, а також як інші компоненти програмної системи взаємодіють між собою в цих процесах.

Діаграма послідовності є способом формалізації сценаріїв варіантів використання. Її перевага полягає в можливості виявити набір компонентів, що взаємодіють, і описати потоки повідомлень між ними на ранніх етапах опису сценарію.

Ці компоненти та потоки повідомлень згодом трансформуються у конкретні класи (об’єкти) та методи цих об’єктів (використовуючи, наприклад, термінологію Java). Таким чином, уже на початковому етапі створюється і уточнюється модель подій системи (дій), які підтримуються й обробляються цими класами (об’єктами).

За допомогою діаграми послідовностей можна зобразити взаємодію елементів моделі у вигляді своєрідної часової осі «життя» всіх об’єктів, залучених до виконання варіанту використання програмної системи, з метою виконання завдання або досягнення певної цілі.Діаграма послідовності показує лише ті об’єкти, які безпосередньо беруть участь у взаємодії; зв’язки з іншими об’єктами не візуалізуються.

У діаграмі послідовності передбачається наявність часової осі, що дозволяє візуалізувати часові зв’язки між переданими повідомленнями.

Лінія життя об’єкта зображається як вертикальна пунктирна лінія, пов’язана з певним об’єктом у діаграмі послідовності. Лінія життя вказує на період, протягом якого об’єкт існує в системі і, отже, може брати участь у взаємодіях.

Ось спрощені визначення:

- **Лінія життя об’єкта**(object lifeline) — вертикальна лінія в діаграмі послідовності, яка представляє існування об’єкта протягом певного періоду;
- **Повідомлення** — сигнал на виконання певної операції, яку має здійснити об’єкт-отримувач.

У UML кожна взаємодія описується через повідомлення, що обмінюються між об’єктами. Повідомлення — це завершена одиниця інформації, що передається від одного об’єкта до іншого. Отримання повідомлення ініціює виконання певних дій.

Якщо об’єкт існує постійно в системі, його лінія життя простягається по всій діаграмі послідовності зверху вниз. Діаграма послідовності має дві виміри:

1. **Горизонтальний вимір** — зліва направо — показує вертикальні лінії, які представляють лінії життя окремих об’єктів, що беруть участь у взаємодії. Об’єкти можуть перебувати в активному стані (виконувати дії) або очікувати повідомлення від інших об’єктів.
Щоб чітко вказати активний стан об’єкта, UML використовує поняття **фокус контролю**. Він зображається як вузький вертикальний прямокутник. Верхня межа прямокутника позначає початок контролю об’єкта, а нижня — його завершення. Прямокутник розташовується під іменем об’єкта та тимчасово замінює лінію життя під час активності об’єкта.

2. **Вертикальний вимір** — це часова вісь, яка йде зверху вниз. Початковий момент часу зображується у верхній частині діаграми. Процес взаємодії між об’єктами відбувається шляхом передачі повідомлень від одного об’єкта до іншого. Повідомлення позначаються горизонтальними стрілками з підписом — назвою повідомлення. Ці повідомлення впорядковані хронологічно: ті, що вище на діаграмі, надсилаються раніше, ніж ті, що нижче. Часова вісь не масштабується; діаграма послідовності моделює лише відносний порядок дій, а не точний час.

На **рисунку 1.3** зображено діаграму послідовності для розроблюваної системи.

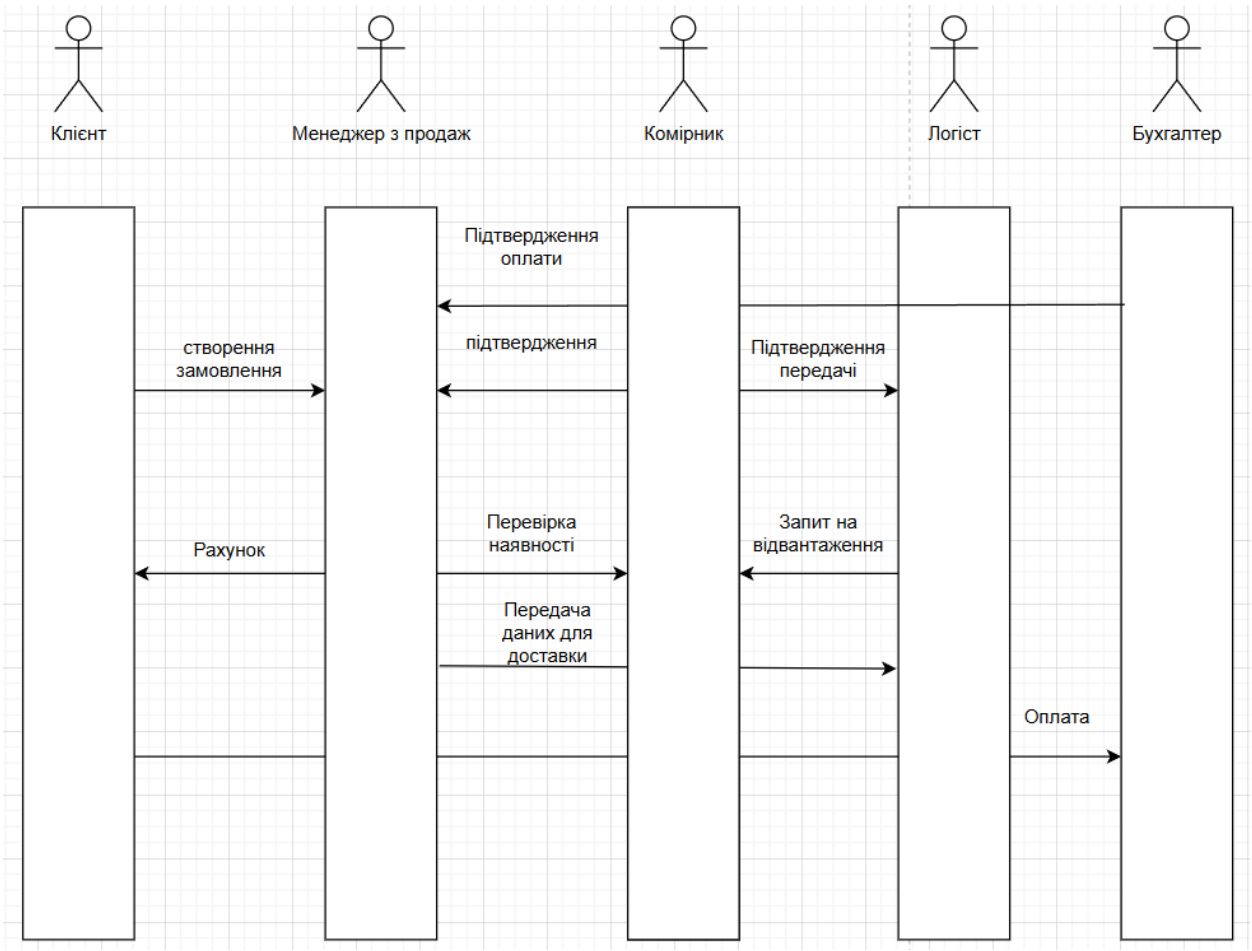


Рис.1.3 Діаграма послідовності

2. ПІДТРИМКА ІНФОРМАЦІЇ СИСТЕМИ

2.1 Загальна інформація про ER-діаграму

Під час процесу розробки програмного забезпечення розробник повинен ретельно вивчити всі вимоги клієнта. Базу даних необхідно спроєктувати так, щоб вона була зрозумілою, точно відображала вимоги та уникала надмірності даних.

Для спрощення процесу розробки (проєктування) баз даних використовуються так звані семантичні моделі даних. Серед різних типів моделей найвідомішою є ER-модель (Entity-Relationship Model).

ER-модель (Entity-relationship model або Entity-relationship diagram) — це семантична модель даних, що полегшує процес проєктування баз даних. Усі типи баз даних — реляційні, ієрархічні, мережеві та об'єктно-орієнтовані — можуть бути побудовані на основі ER-моделі. Основні поняття цієї моделі — це «сутність», «зв'язок» і «атрибут».

Під час створення великих баз даних ER-модель допомагає уникнути помилок проєктування, які вкрай складно виправити, особливо коли база даних вже використовується або перебуває на стадії тестування. Помилки в структурі бази можуть призвести до необхідності переробки програмного коду, який керує цією базою. Як наслідок, можуть бути неефективно використані час, кошти та людські ресурси.

ER-модель представляє базу даних у вигляді зрозумілих графічних діаграм. Вона візуалізує процес, що описує певну предметну область. ER-діаграма графічно відображає сутності, атрибути й зв'язки між ними.

Проте ER-модель — це лише концептуальний рівень моделювання. Вона не включає деталі реалізації. Для однієї й тієї самої ER-моделі реалізація може суттєво відрізнятися.

Сутність у базі даних — це будь-який об'єкт, який можна чітко визначити в межах предметної області. Розробник повинен коректно ідентифікувати сутності.

В ER-моделі існують такі типи сутностей:

- **слабка сутність** – залежить від сильної сутності;

- **сильна сутність** – не залежить від інших сутностей.

Кожен тип сутності має набір атрибутів, що описують її.

Типи атрибутів:

- **рості атрибути** – складаються з одного елемента і можуть бути частиною складених атрибутів (напр., код книги в бібліотеці або курс студента);
- **складені атрибути** – складаються з кількох простих (напр., адреса: країна, місто, вулиця, номер будинку);
- **атрибути з єдиним значенням** – мають лише одне значення для сутності (напр., номер залікової книжки студента);
- **багатозначні атрибути** – можуть містити кілька значень (напр., телефон працівника: мобільний, домашній тощо);
- **похідні атрибути** – значення обчислюється на основі інших атрибутів (напр., поточний навчальний рік студента — від дати вступу).

Для створення прикладних програм можна використовувати різні підходи до представлення типів сутностей і атрибутів.

Можна обрати перевірену технологію джерела даних (наприклад, Microsoft SQL Server, Microsoft Access), яка є стандартизованою і має повний інструментарій підтримки СУБД. Або ж розробити власний формат бази даних з обробкою даних через спеціальні команди (наприклад, імпорт/експорт).

Також можливий гібридний підхід. Сучасні інструменти розробки містять бібліотеки для обробки та візуалізації даних (колекції, масиви, компоненти тощо).

У реляційній СУБД (Microsoft Access, Microsoft SQL Server) сутності — це таблиці, а атрибути — поля таблиць. Запис у таблиці — це екземпляр сутності.

Типи атрибутів реалізуються так:

- **Прості та з єдиним значенням** — базові типи (int, integer);
- **З плаваючою крапкою** — float, double;

- **Текстові** — string;
- **Складені** — об’єкти з вкладеними простими атрибутами;
- **Багатозначні** — масив або колекція;
- **Похідні** — обчислюване поле;
- **Первинний ключ** — унікальне поле (часто типу int).

У власному форматі типи сутностей відображаються як класи або структури, атрибути — як поля, а методи виконують обробку даних. Зв’язки реалізуються через інтерфейси або патерни проектування

2.2 Побудова ER-діаграми

ER-діаграма була створена за допомогою засобу моделювання баз даних **ERWin Data Modeler**.

CA ERWin Data Modeler (також відомий як AllFusion ERWin Data Modeler) — це програмне рішення для проєктування, впровадження та підтримки баз даних і сховищ даних (Data Warehouses).

Можливості ERWin:

- візуалізація складних структур даних;
- відстеження інформаційних ресурсів;
- дотримання корпоративних стандартів керування даними;
- автоматизація проєктування й синхронізація моделі з реальною базою.

ERWin підтримує моделювання як бізнес-процесів, так і структур даних.

У ERwin версії 7 можна моделювати як дані, так і процеси. Для цього в ERwin є відповідні програмні модулі. Модуль ERwin Data Modeller призначений для моделювання даних, а модуль ERwin Process Modeller - для моделювання процесів.

ERwin Process Modeller дозволяє здійснювати функціональне моделювання. За допомогою ERwin Process Modeller можна створювати функціональні діаграми, діаграми робочих процесів і діаграми потоків даних.У пізніших версіях модуль процесів було видалено. З версії 8 ERWin фокусується лише на моделюванні даних.

Основна ціль — створення моделей високого рівня, генерація схем баз даних і описів на рівні коду.

ERWin підтримує:

- **логічні моделі** (використовують ER-діаграми);
- **фізичні моделі** (відображають реальну реалізацію бази).

Переваги:

- шаблони та стандарти;
- автоматичне порівняння та синхронізація моделі й бази;
- інтеграція з UML-нотаціями та іншими інструментами.

На рисунку 2.4 зображено діагарама класів

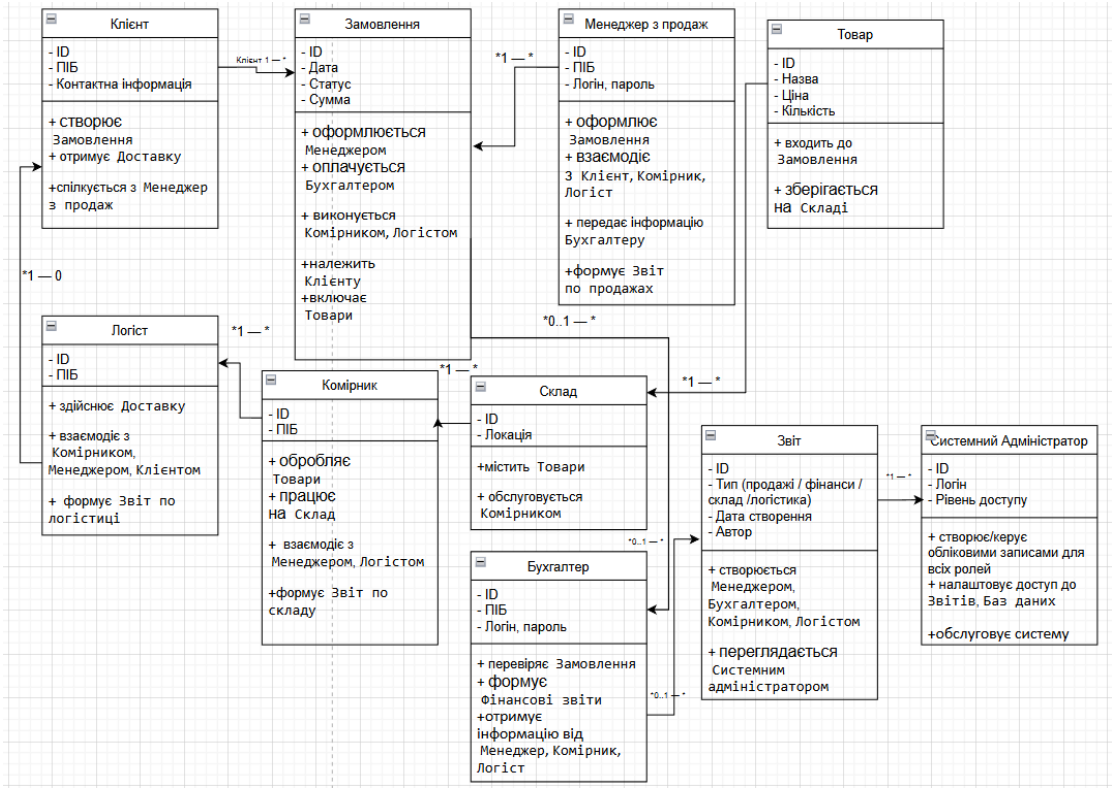


рис. 2.1 Діагарама класів

На рисунках 2.2 – 2.5 наведено діаграму класів для окремих модулів

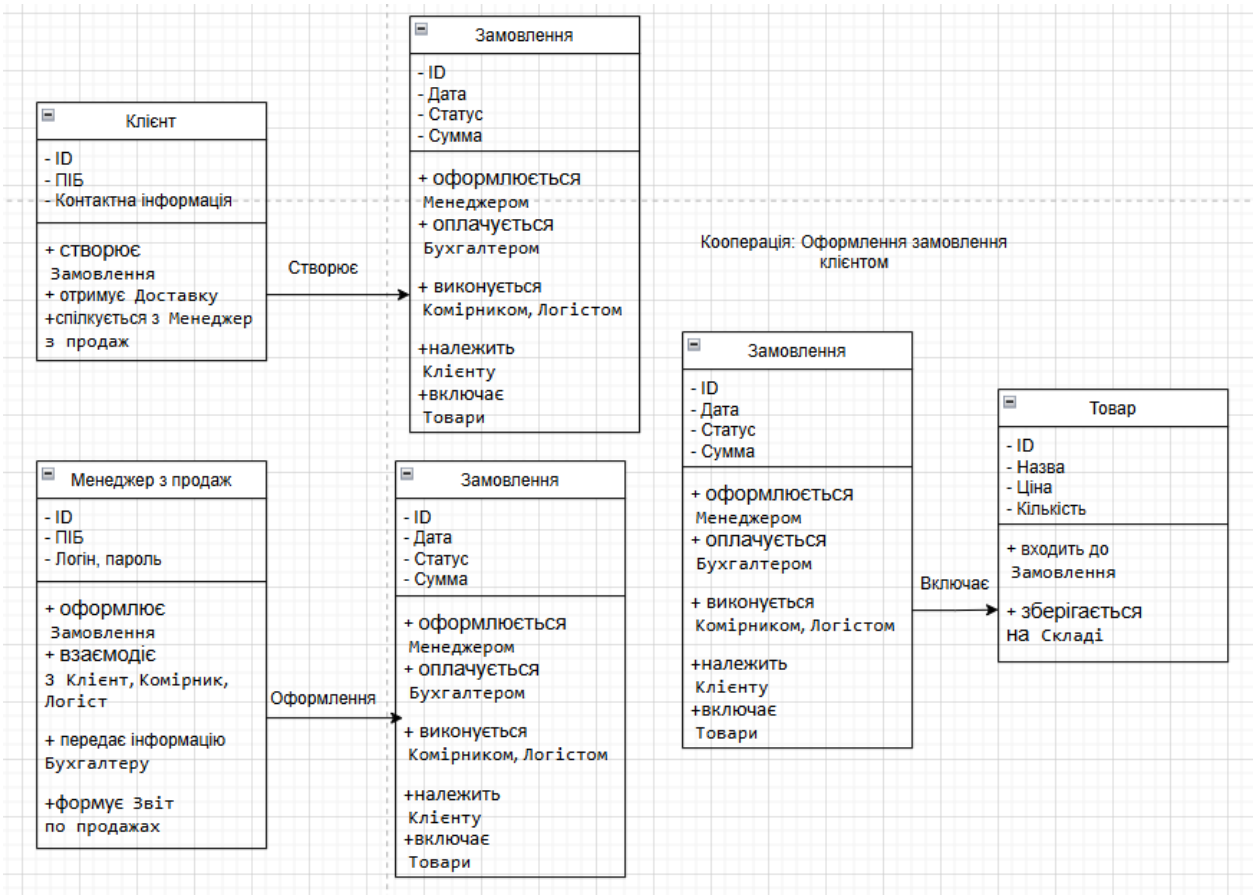


Рис.2.2 Оформлення замовлення клієнтом

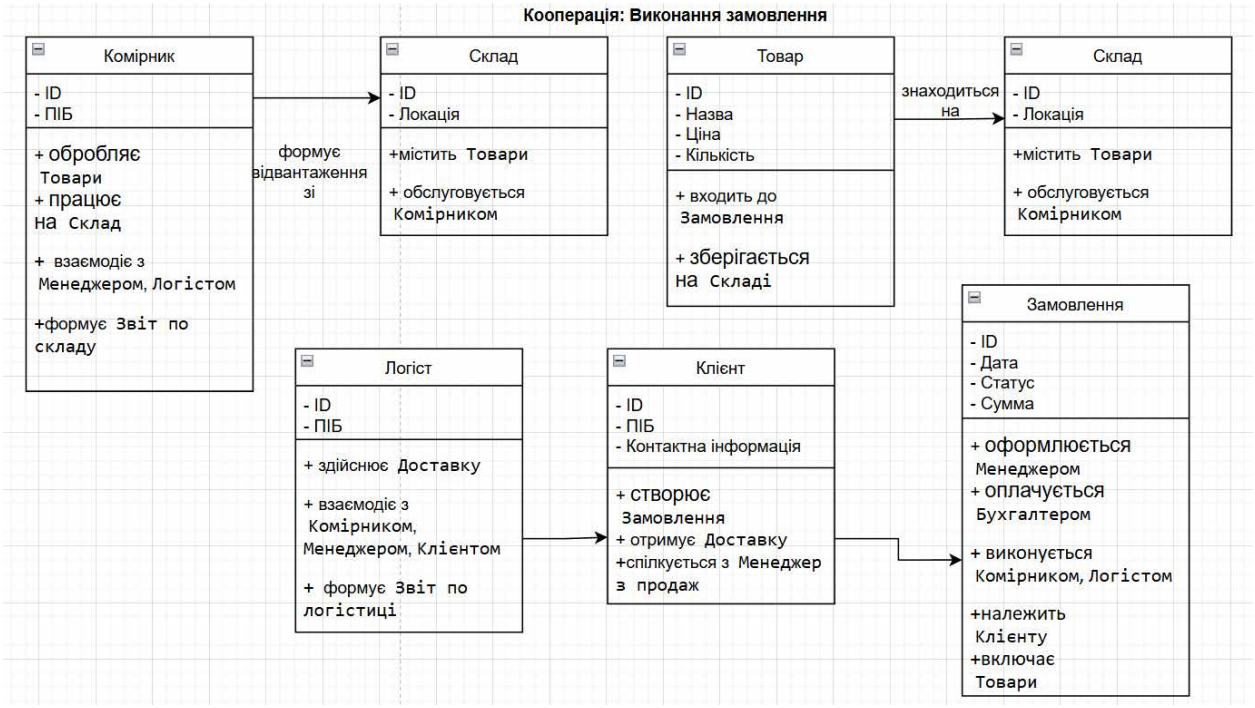


Рис.2.3 Виконання замовлення

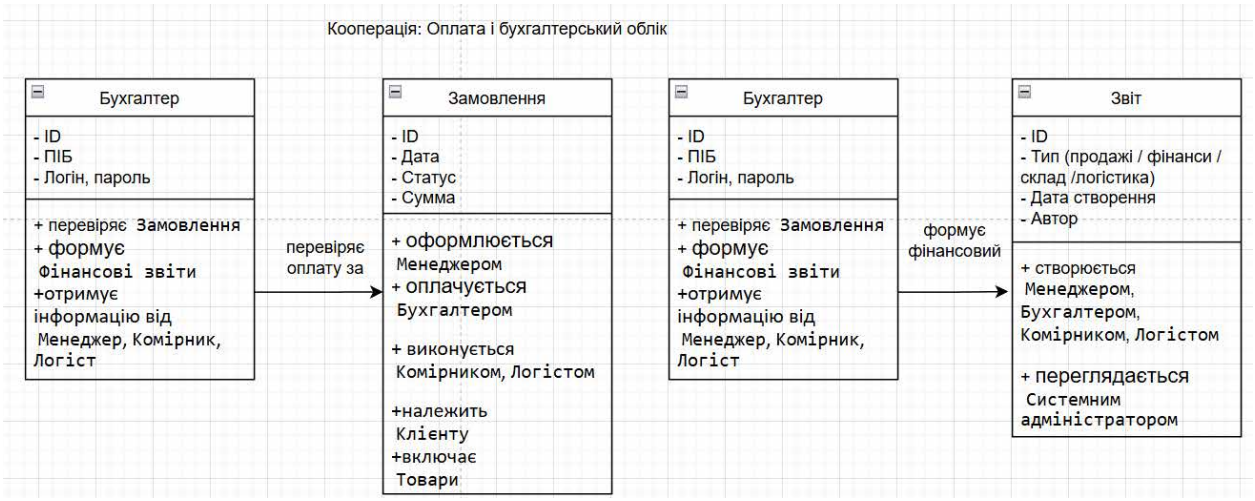


Рис.2.4 Оплата і бухгалтерський облік

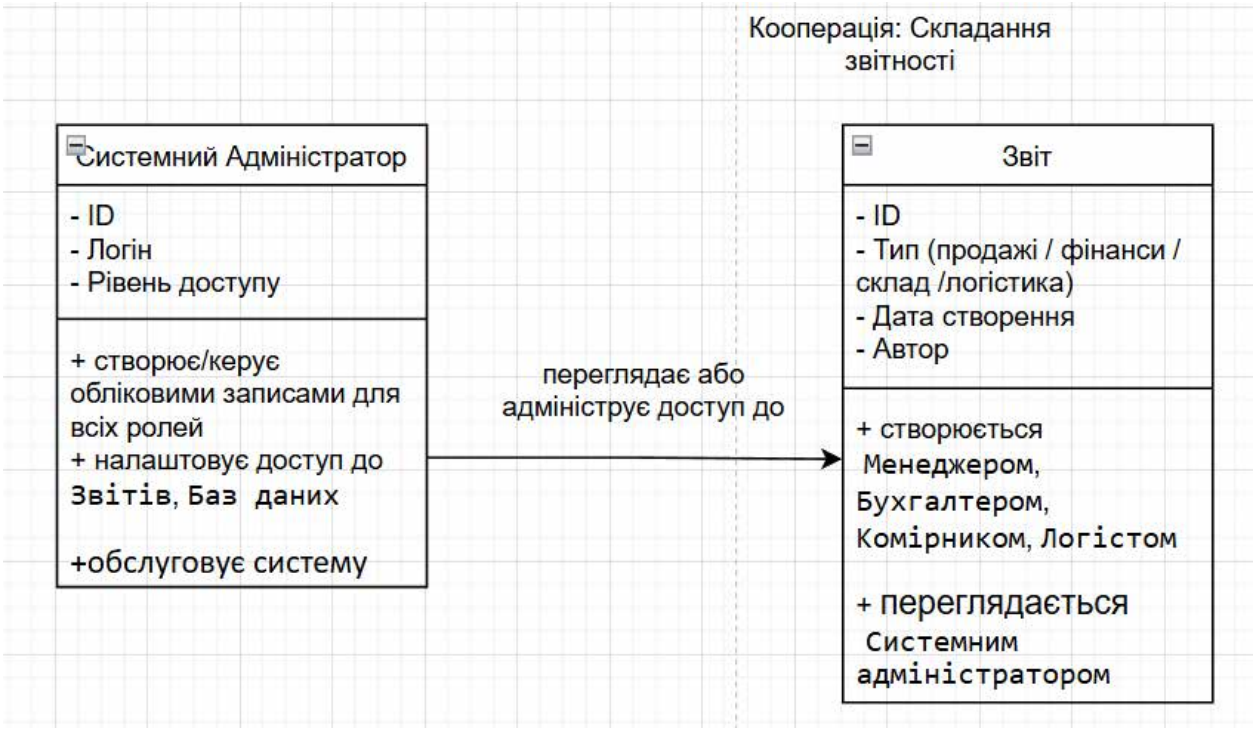


Рис.2.5 Складання звітності

2.3 Вибір та обґрунтування СУБД

Однією з основ сучасних ІТ є бази даних — структуровані, логічно пов’язані набори даних, пов’язані з конкретним завданням, разом із системою керування базами даних (СУБД).

СУБД забезпечує обробку, зберігання, доступ та організацію даних на запит користувача чи системи.

Вибір СУБД залежить від:

- **масштабованості** — здатності підтримувати більші обсяги даних та користувачів;
- **продуктивності** — швидкості обробки запитів;
- **інтегрованості та безпеки** — захисту від несанкціонованого доступу.
- **підтримки транзакцій** — властивостей ACID;
- **інтеграційних можливостей** — з іншими платформами;
- **вартості** — наявності безкоштовних або ліцензованих версій.

Було обрано **Microsoft SQL Server**, оскільки він:

- забезпечує високу продуктивність і надійність;
- має зручні графічні інструменти для проєктування;
- інтегрується з Visual Studio і .NET;
- пропонує засоби безпеки: рольовий доступ, шифрування;
- має засоби резервного копіювання та відновлення;
- підтримує збережені процедури, тригери та бізнес-логіку;
- має безкоштовну версію SQL Server Express для малих і середніх застосунків.

Цей вибір забезпечує масштабованість, простоту обслуговування та надійну підтримку.

2.4 Створення БД

Для функціонування бази даних було створено:

- додавання даних до таблиць;
- додавання користувачів;

- створення тригерів;
- створення уявлень;
- створення процедур.

Після того, як база даних створена, створюються таблиці бази даних .

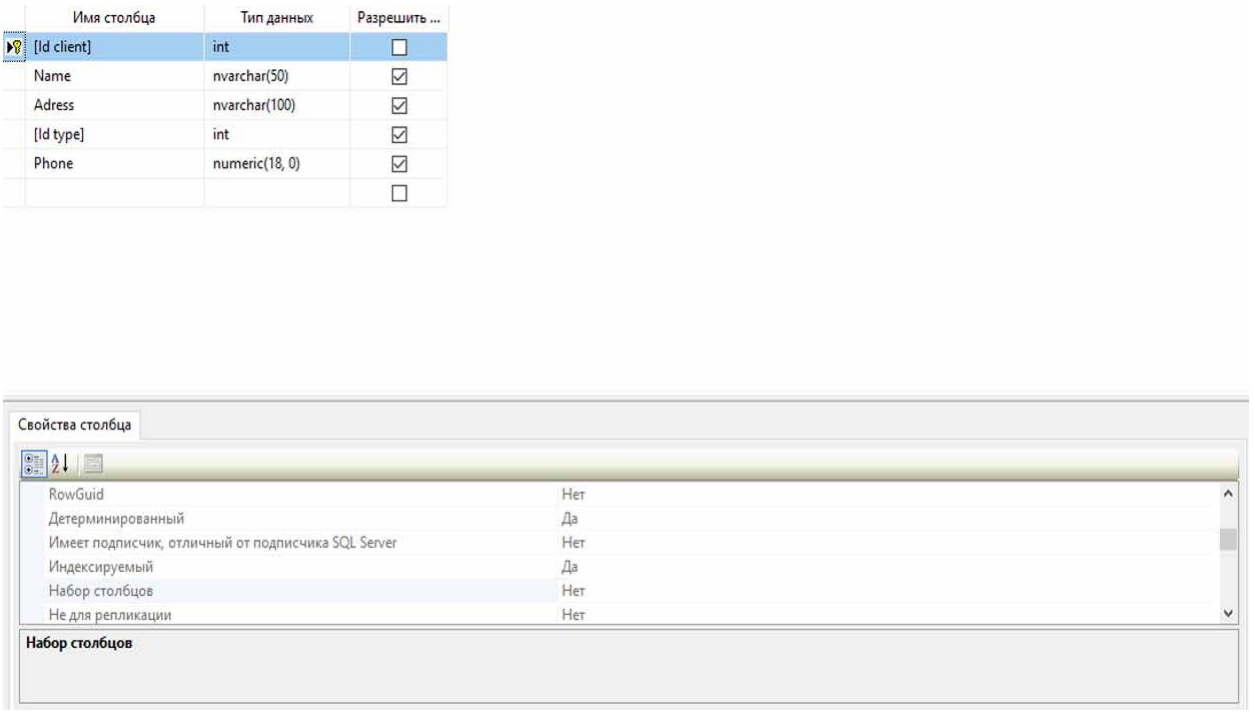


Рис. 2.6 Створення таблиці через «конструктор»

Після того, як визначили поля (атрибутів) таблиці, повинні зберегти таблицю, вказавши ім'я таблиці.

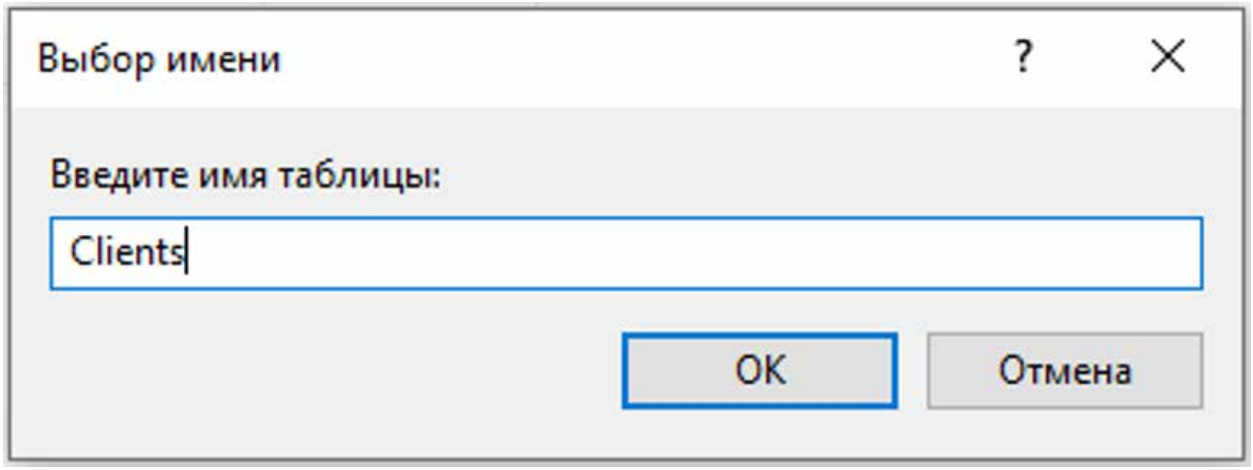


Рис. 2.7 Збереження даних таблиці

2.5 Додавання користувачів до БД

Створюємо користувача бази даних за допомогою конструктора (рис. 2.8).

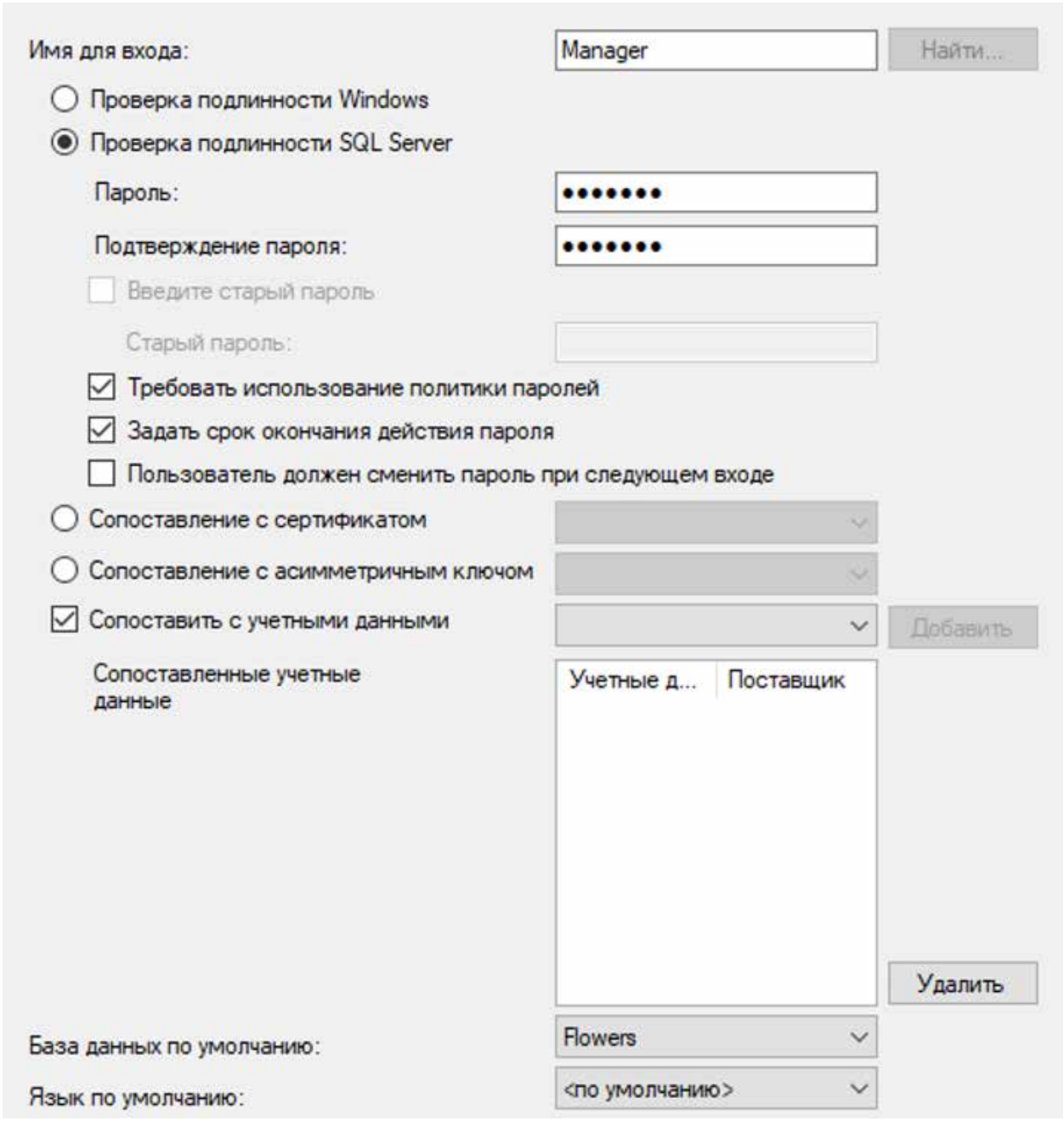


Рис. 2.8 Створення користувача

3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Вибір інструментів розробки

При розробці програмного забезпечення стояв вибір між C++ та C#. Обидві мови програмування були розглянуті з урахуванням їхніх переваг і недоліків.

Мова C++

C++ — Це мова з вражаючими можливостями, Яка забезпечує всі необхідні засоби для створення ефективного програмного забезпечення будь-якого рівня— від низькорівневих утиліт і драйверів до комплексних програмних систем різного виду.

Переваги C++:

- підтримує кілька стилів і технологій програмування, зокрема традиційне імперативне програмування та об’єктно-орієнтоване;
- дозволяє працювати з пам’яттю, адресами та портами на низькому рівні;
- дозволяє створювати загальні (генеричні) алгоритми для різних типів даних, включно зі спеціалізацією та обчисленнями на етапі компіляції за допомогою шаблонів (templates);
- кросплатформеність: існують компілятори для різних платформ;
- ефективність: мова дає повний контроль над виконанням, забезпечуючи максимальну продуктивність.

Недоліки C++:

Багато недоліків успадковано від мови C через потребу максимальної сумісності з нею:

- минтаксис, що часто призводить до помилок;
- макроси, які потенційно небезпечні, хоч і менш необхідні через шаблони та inline-функції. Стандартні бібліотеки часто містять небезпечні макроси;
- деякі перетворення типів неінтуїтивні (наприклад, операції між знаковими і беззнаковими цілими числами повертають беззнаковий результат);
- C++ є складною та багатослівною мовою, що ускладнює її вивчення і розробку компіляторів;
- поєднує всі конструкції C з новими, що спричиняє дублювання;
- множинне успадкування в об’єктно-орієнтованому програмуванні викликає логічні проблеми та ускладнює реалізацію;
- шаблони можуть генерувати велику кількість коду, а часткова спеціалізація погано підтримується компіляторами.

Мова C#

C# — об’єктно-орієнтована мова програмування, розроблена між 1998 і 2001 роками командою на чолі з Андерсом Хейлсбергом у компанії Microsoft як основна мова для платформи .NET.

C# належить до родини мов із синтаксисом стилю C, подібного до C++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів, атрибути, події, властивості, узагальнення та XML-коментарі. Вона успадкувала особливості з C++, Java, Delphi, Modula і Smalltalk, виключивши

проблемні модулі (наприклад, не підтримує множинне успадкування класів на відміну від C++).

Переваги C#:

- сумісність з більшістю продуктів Microsoft;
- багато інструментів безкоштовні для малих компаній і окремих розробників (Visual Studio, Windows Server, Parallels Desktop for Mac Pro тощо);
- типи даних мають фіксовані розміри (наприклад, int — 32 біти, long — 64 біти), що підвищує «портативність» мови і спрощує розробку;
- автоматичне збирання сміття знімає потребу в ручному управлінні пам’яттю у більшості випадків;
- вбудовані конструкції допомагають розуміти та писати код (не впливаючи на компіляцію);
- низький поріг входу: знайомий синтаксис, зручна для початківців;
- завдяки Xamarin на C# можна розробляти додатки для iOS, Android, macOS і Linux.

Недоліки C#:

- брієнтована переважно на платформу Windows.
- безкоштовна лише для малих компаній, окремих розробників і студентів.

Висновок:

У порівнянні обох мов C# краще підходить для розробки прикладних програм.

Інструменти розробки

Для створення прикладного програмного забезпечення було використано **Microsoft Visual Studio**.

Для розробки прикладного ПЗ використовувалося Microsoft Visual Studio. Microsoft випустила в рамках концепції .NET Framework продукт Visual Studio

.NET Enterprise Architect 2003, що реалізував останні досягнення у сфері програмування — особливо візуальні методи розробки.

Visual Studio .NET — це повна багатомовна середовище розробки для платформи Microsoft .NET. Воно пропонує технології, що спрощують створення, розгортання і розширення безпечних, масштабованих і високодоступних веб-застосунків і XML-вебсервісів.

Microsoft Visual Studio включає інтегроване середовище розробки (IDE) і різні допоміжні інструменти. Підтримує різні типи застосунків (консольні, Windows, мобільні та веб-застосунки) і кілька мов програмування, включно з C++ і C#.

Visual Studio — це повністю інтегроване середовище, яке робить процес написання, відлагодження та компіляції коду максимально простим. Фактично це складний багатодокументний додаток, за допомогою якого можна виконувати майже будь-яке завдання розробки.

Середовище Visual Studio є одним з найпопулярніших IDE. Нові версії виходять щороку або раз на два роки.

У цьому середовищі проєктування користувацьких інтерфейсів і додавання компонентів відбувається через дизайнер форм. Всі компоненти доступні в палітрі компонентів. Вікно властивостей не є постійно відкритим — воно відкривається за потребою. Інтерфейс і описи інструментів англійською мовою.

Інтерфейс Visual Studio інтуїтивно зрозумілий. Переваги — широкий супровід, багато онлайн-уроків і регулярні оновлення від Microsoft.

У середовищі є більшість необхідних бібліотек, які можна підключати за потребою. Microsoft Visual Studio — професійне середовище розробки, що дозволяє створювати повнофункціональні застосунки.

Особливості Visual Studio:

1. **текстовий редактор**

Дозволяє писати код на C#, Visual Basic і C++. Має автоматичне відступлення, форматування, парні дужки, підсвічування синтаксису. Синтаксис перевіряється в реальному часі, помилки підсвічуються під час введення — це називається відлагодження на рівні дизайну (Design-time debugging). Редактор містить IntelliSense — автоматичне пропонування імен класів, полів, методів і параметрів;

2. **візуальний дизайнер форм**

Дозволяє додавати елементи UI і елементи доступу до даних. Visual Studio автоматично генерує потрібний код C# для створення цих компонентів (усі елементи керування .NET є екземплярами базових класів);

3. **допоміжні вікна**

Ці вікна надають можливість переглядати та редагувати проект, зокрема класи у вихідному коді, а також властивості з їхніми початковими значеннями, доступні для класів Windows Forms і Web Forms. Крім того, вони можуть використовуватись для налаштування параметрів компіляції;

4. **інтегрована компіляція**

Замість запуску компілятора C# з командного рядка компіляція відбувається безпосередньо через меню IDE. Visual Studio керує всіма параметрами компіляції, включно з посиланнями та форматом вихідних файлів (наприклад, виконуваний файл або DLL). Також може автоматично запускати скомпільовану програму для тестування;

5. **інтегрований зневадник (дебагер)**

Оскільки код рідко працює з першого разу, Visual Studio пропонує безшовну інтеграцію дебаггера з точками зупину і відслідковуванням змінних без виходу з IDE;

6. **інтеграція зовнішніми інструментами**

Visual Studio можна інтегрувати з іншими інструментами, має доступ до баз даних, перегляду таблиць у SQL Server та веб-навігацію через вбудований Internet Explorer;

7. **інтегрована довідка MSDN**

Вся довідкова інформація по технологіях Microsoft зібрана в одному продукті — MSDN (Microsoft Developer Network). Visual Studio забезпечує прямий доступ до документації MSDN всередині IDE. Наприклад, у редакторі можна виділити ключове слово і натиснути <F1>, щоб отримати релевантну інформацію. Аналогічно можна отримати допомогу по помилках компіляції.

Visual Studio також містить графічні редактори і XML-дизайнери, підтримує розробку Windows-додатків для мобільних пристроїв, Microsoft Office, Windows Workflow Foundation і візуальні інструменти для проєктування класів.

Хоча Microsoft Visual Studio — лише одне з декількох середовищ розробки для C#, воно має значні переваги і є єдиним повнофункціональним середовищем.

Безкоштовна версія — Microsoft Visual Studio Express — має деякі функціональні обмеження, однак для початківців більш ніж достатньо.

Завантажити Visual Studio можна тут: https://visualstudio.microsoft.com/ru/

Основні версії:

- -Community;
- -Professional;
- -Enterprise.

Visual Studio Community

- **безкоштовна версія** для індивідуальних розробників, студентів, відкритого програмного забезпечення і невеликих команд;
- підтримує розробку додатків для Windows, Android, iOS, веб і хмарних сервісів;
- містить більшість основних функцій для розробки, налагодження і тестування;
- ідеально підходить для навчання, хобі та малих проектів.

Visual Studio Professional

- **платна версія** з розширеними можливостями, орієнтована на професійних розробників і невеликі команди;
- включає покращені інструменти для співпраці, управління кодом, тестування і продуктивності;
- містить підтримку інтеграції з Azure, можливості для розробки корпоративних застосунків;
- забезпечує технічну підтримку від Microsoft.

Visual Studio Enterprise

- **найповніша та найпотужніша версія** для великих команд і складних проектів;
- має всі функції Professional, а також додаткові інструменти для архітектури, тестування продуктивності, автоматичного тестування, аналізу коду та DevOps;
- підтримує розробку масштабованих корпоративних рішень і складних систем;

- включає розширену підтримку та доступ до ексклюзивних сервісів Microsoft.

Корисні розширення для Visual Studio:

- **ReSharper** — статичний аналізатор коду (платний, дуже корисний);
- **Web Essentials** — прискорює розробку UI;
- **Productivity Power Tools 2017/2019** — набір міні-розширень для комфортної розробки;
- **GitHub Extension** — інтеграція з репозиторіями;
- **Visual Studio Spell Checker** — допомагає виправляти орфографічні помилки в коді, працює подібно до Word, підкреслюючи помилки червоним. Розпізнає різні конвенції іменування;
- **IntelliSense** — одна з найважливіших функцій середовища, яка дозволяє автозаповнення класів, методів і параметрів, покращуючи якість і швидкість розробки.

3.2 Принципи роботи в середовищі розробки візуального програмування

Microsoft Visual Studio надає користувачам можливість виконувати різноманітні завдання під час розробки програмного забезпечення: створювати графічний інтерфейс користувача, редагувати вихідний код, компілювати елементи застосунку та виконувати покрокове налагодження.

Середовище розробки застосунків Microsoft Visual Studio

Microsoft Visual Studio дозволяє користувачам:

- Проектувати графічні інтерфейси;
- Редагувати програмний код;
- Компілювати компоненти застосунку;

- Покроково налагоджувати застосунки.

Елементи графічного інтерфейсу Microsoft Visual Studio є типовими для програм Windows. Розміри й розташування вікон у межах інтерфейсу залежать від конкретної конфігурації системи. Користувачі можуть змінювати розміри та позицію окремих елементів або згортати їх, щоб звільнити місце для інших, необхідних на даний момент.

Основні елементи вікна Microsoft Visual Studio включають:

- **Меню вікна та стандартна панель інструментів** – містять команди для роботи в середовищі Visual Studio;
- **Провідник рішень** – відображає проєкти та файли поточного рішення.
- **Панель інструментів (Toolbox)** – показує компоненти, що використовуються в проєктах Visual Basic;
- **Вікно властивостей** – відображає та дозволяє змінювати властивості об'єктів рішення, включно з формами та вбудованими елементами.

Середовище Microsoft Visual Studio ґрунтується на **.NET Framework** — технології, створеній для розробки як настільних, так і вебзастосунків.

Платформа .NET Framework складається з двох ключових компонентів:

- **Common Language Runtime (CLR)** – керує виконанням коду програми, забезпечуючи керування пам’яттю та підтримку віддаленого виконання;
- **Бібліотека класів** – об’єктно-орієнтована колекція типів, що може використовуватись у розробці застосунків — від традиційних консольних до застосунків з графічним інтерфейсом та вебформами.

Клас визначає тип об’єкта; об’єкт — це екземпляр певного класу. За допомогою **просторів імен (namespaces)** класи можуть організовуватися ієрархічно. Простори імен допомагають категоризувати назви класів і уникати

конфліктів назв. Простір імен може містити класи, перерахування, структури, інтерфейси та інші простори імен.

Формат позначення простору імен зазвичай такий:

NamespaceName.TypeName

Типові простори імен у щойно створеному проєкті Visual Studio:

- **System** – містить базові класи для визначення типів даних, необхідних у будь-якому застосунку;
- **System.Windows.Forms** – містить класи для реалізації поведінки віконних форм, які є основою застосунків Microsoft Windows;
- **System.Drawing** – надає доступ до графічних інтерфейсів пристроїв; класи використовуються для малювання ліній, зображень та інших графічних елементів у вікнах застосунку;
- **System.Collections** – включає класи для роботи з масивами, списками, словниками тощо;
- **System.Data** – використовується в застосунках, що працюють з базами даних.

Рішення та проєкти

Рішення та **проєкти** — це контейнери, які Microsoft Visual Studio використовує для зберігання й організації коду, написаного у вбудованому середовищі розробки.

- **Контейнери** — це об'єкти, які містять інші об'єкти;
- **Рішення** — це віртуальний контейнер, що групує властивості та компоненти, пов’язані з певним проєктом;
- Рішення не обробляються компілятором (який перетворює високорівневий код на машинний); вони слугують контейнерами вищого рівня для інших елементів.

Самі по собі рішення не виконують жодної функції, окрім зберігання інших елементів. **Проєкти** є основними одиницями, які можна розміщувати в рішенні.

У Microsoft Visual Studio в одному вікні може бути відкрито лише одне рішення. Щоб працювати з кількома рішеннями одночасно, потрібно запустити кілька копій Visual Studio.

Створення нового проєкту

Щоб створити новий проєкт, використовуйте один із таких способів:

- На стартовій сторінці натисніть гіперпосилання **Створити: Проєкт**;
- У меню **Файл** оберіть **Створити → Проєкт**;
- Натисніть кнопку **Створити проєкт** на стандартній панелі інструментів.

Це відкриє діалогове вікно **Створення проєкту**, де необхідно вибрати тип проєкту, вказати його назву та місце зберігання.

Створення проєкту створює нову папку проєкту на диску комп’ютера разом із файлом рішення, який має вказану назву, та посиланням на проєкт.

4. ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Тестування системи

Під час розробки програмного забезпечення розробники дуже часто стикаються з різними проблемами. Іноді програмний продукт створений неякісно, а іноді витрати на розробку та тестування значно перевищують запланований бюджет. Буває й так, що проблеми виникають одночасно. Незважаючи на значні витрати на тестування, досягти високої якості програмного забезпечення вдається не завжди. Для багатьох компаній і підприємств низька якість ПЗ — що, на жаль, іноді трапляється — призводить не лише до краху або занепаду бізнесу, а щонайменше до втрати репутації та значних фінансових збитків.Питання, що таке саме є тестування, не таке просте, як може здатися. Різні джерела надають різні визначення тестування.

Згідно зі стандартом IEEE Std 829-1983, тестування — це процес аналізу програмного забезпечення з метою виявлення відмінностей між реальними та очікуваними властивостями, а також для оцінки характеристик ПЗ. Ці відмінності називаються дефектами.

Тестування зосереджене головним чином на оцінюванні якості програмного продукту за допомогою таких методів, як:

- виявлення та документування дефектів якості;

- надання загальних рекомендацій щодо якості;
- перевірка відповідності основним вимогам продукту на основі конкретних прикладів;
- перевірка правильності функціонування продукту згідно з його проєктом;
- перевірка правильного виконання вимог.

Відповідно до ГОСТ Р ІСО/МЕК 12207-99, життєвий цикл ПЗ включає, серед іншого, процеси верифікації, валідації, колективного аналізу та аудиту.

Верифікація — це процес визначення того, чи працює програмний продукт повністю відповідно до вимог або умов, встановлених на попередніх етапах розробки. Цей процес може включати аналіз, інспекцію та тестування.

Необхідно визначити інструменти, які будуть використовуватись (якщо такі є) для пошуку та документування виявлених дефектів і недоліків. Якщо тестування враховується ще на етапі початку розробки, перевірка продукту не призведе до небажаних результатів чи неприємних сюрпризів. Відповідно, якість програмного продукту, ймовірно, буде досить високою.

Валідація — це процес визначення, чи відповідають встановлені вимоги до системи або програмного продукту його передбаченому функціональному призначенню.

Колективний аналіз — це процес оцінювання стану та, за потреби, перевірки результатів роботи (продуктів) проєкту.

Аудит — це процес визначення відповідності вимогам, планам і умовам контракту на розробку.

Ці процеси разом утворюють те, що зазвичай називають тестуванням. При реалізації проєкту необхідно враховувати, за якими стандартами та вимогами буде проводитися тестування програмного продукту.

Рівні тестування програмного забезпечення

Тестування зазвичай проводиться під час етапів розробки та супроводу на кількох рівнях. Рівень тестування визначає, **що саме** перевіряється на даному етапі: окремий модуль, група модулів чи вся система. Жоден рівень тестування не є важливішим за інші — усі вони критично важливі, незалежно від використовуваних моделей і методів.

Зазвичай розрізняють такі типи тестування:

- модульне тестування (unit testing);
- інтеграційне тестування;
- системне тестування.

Кожен тип охоплює різні аспекти виявлення помилок.

Модульне тестування

Цей рівень дозволяє перевірити правильність роботи окремих компонентів системи. Що саме є компонентом, залежить від контексту. Цей тип тестування детально описаний у стандарті IEEE 1008-87 "Standard for Software Unit Testing", який визначає систематичний і документований підхід до модульного тестування.

Модульне тестування дозволяє перевіряти окремі модулі, методи, функції та класи. Такий підхід дозволяє з’ясувати, чи правильно працюють окремі частини коду, та швидко перевірити, чи зміни у коді впливають на функціональність. Після перевірки функцій і операцій спеціаліст з контролю якості переходить до наступного етапу — інтеграційного тестування.

Інтеграційне тестування

Цей рівень — це процес перевірки взаємодії між компонентами ПЗ або модулями програми. Воно зазвичай виконується після модульного тестування. На цьому етапі аналізуються окремі частини коду та їх взаємодія.

Класичні стратегії інтеграційного тестування — «згори донизу» і «знизу вгору» — застосовуються до традиційних ієрархічно структурованих систем, але

важко реалізуються у слабо зв’язаних системах на основі сервіс-орієнтованої архітектури (SOA).

Сучасні стратегії тестування більше залежать від архітектури системи і базуються на виявленні функціональних «потоків» (наприклад, потоків операцій або даних).

Інтеграційне тестування відбувається на досить високому рівні абстракції. Успішне інтеграційне тестування допомагає уникнути проблем, пов’язаних із локальними перевірками або перевіркою після тривалого циклу розробки, коли виявлені дефекти можуть вимагати суттєвої перебудови коду (негативний досвід подібної практики називається «big bang»).

Системне тестування

Системне тестування охоплює всю систему загалом. Більшість функціональних помилок мають бути виявлені на попередніх етапах — модульного та інтеграційного тестування. У системному тестуванні більше уваги приділяється не функціональним вимогам, а питанням безпеки, продуктивності, точності, надійності тощо.

На цьому рівні також перевіряються інтерфейси з іншими застосунками, обладнанням, середовищами.

Системне тестування — це, по суті, завершальний етап розробки програмного продукту. Воно дозволяє оцінити повністю зібраний продукт з точки зору користувача. На цьому етапі виявляються недоліки інтерфейсу, покращується зручність і ергономіка.

Інші методи тестування

Існують також методи «тестування білої скриньки» та «тестування чорної скриньки». При тестуванні білої скриньки тестувальник має повний доступ до вихідного коду та може використовувати власний код для перевірки і виявлення помилок розробника.

При тестуванні чорної скриньки доступ до коду неможливий — перевірка відбувається лише через інтерфейси. Ці типи тестування відрізняються за ступенем обізнаності з внутрішньою структурою системи.

Завершення життєвого циклу

Тестування виконується після проєктування та розробки документації системи. Якщо не виявлено проблемних ділянок, цей етап може стати фінальним у життєвому циклі розробки ПЗ; інакше процес програмування має бути відновлений. Перше тестування майже завжди дає, так би мовити, негативний зворотний зв’язок, оскільки в будь-якій розробці присутні вузькі місця, деякі недоліки та іноді досить серйозні помилки, які неможливо виявити під час створення ПЗ.

4.2 Апаратура та технічні засоби

Склад обчислювальної системи називається її конфігурацією. Конфігурація поділяється на програмну та апаратну частини.

Усі сучасні комп’ютери мають модульну конструкцію. Узгодження функцій між окремими модулями здійснюється за допомогою спеціальних пристроїв, які називаються апаратними інтерфейсами. Стандарти для апаратних інтерфейсів називаються протоколами. Апаратні інтерфейси поділяються на послідовні та паралельні. Передача даних здійснюється через порти.

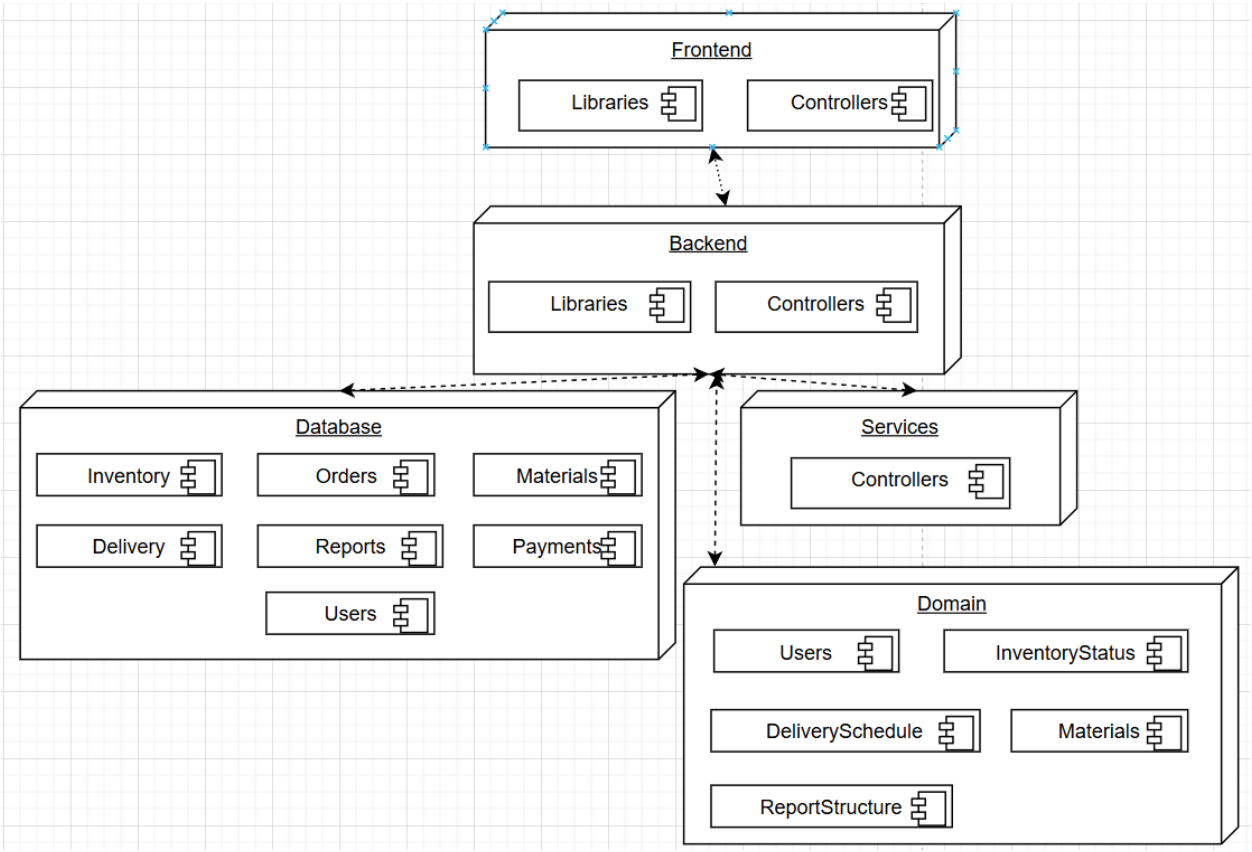
Послідовний інтерфейс забезпечує передачу даних по черзі, біт за бітом, і тому має низьку швидкість передачі. Через це він використовується для підключення так званих "повільних" пристроїв, а також у випадках, коли немає жорстких обмежень за часом обміну даними — наприклад, для підключення клавіатури та миші. Швидкість передачі в послідовних інтерфейсах вимірюється в бітах за секунду.

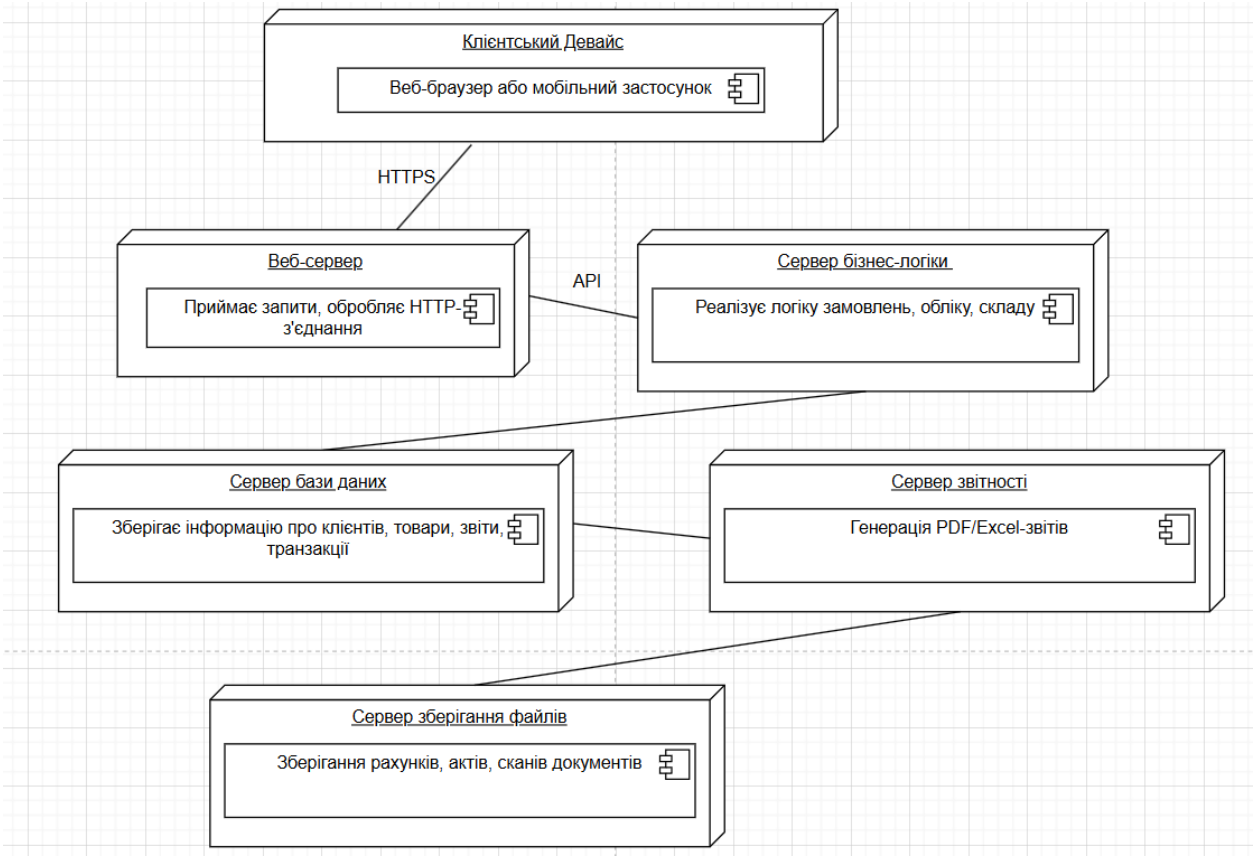
Паралельні інтерфейси забезпечують одночасну передачу даних групами бітів, що суттєво підвищує швидкість передачі. Кількість бітів у групі називається розрядністю інтерфейсу. Існують інтерфейси на 8, 16, 32 та 64 біти.

Вони мають складнішу структуру, ніж послідовні інтерфейси. Застосовуються там, де важлива висока швидкість передачі: для підключення різних пристроїв друку, пристроїв введення та виведення графічних даних, для запису на зовнішні носії тощо. Продуктивність паралельних інтерфейсів вимірюється в байтах за секунду.

Діаграма розгортання системи представлена на рис 4.1.

Для функціонування інформаційної системи необхідні сервер бази даних та комп'ютери. База даних розміщується на сервері, до якого підключатимуться клієнтські комп'ютери. На комп'ютери буде встановлено відповідне програмне забезпечення. Вимоги до апаратного та програмного забезпечення для сервера наведені в Таблицях 2 та 3.





Апаратні вимоги для сервера

Таблиця 2.

Ресурс	**Мінімальний**	**Рекомендований**
Процесор	1.5 ГГц	2 ГГц і вище
Оперативна пам’ять	1 гб	2 Гб і вище
Жорсткий диск	40 Гб	120 Гб
Привод CD (DVD)-ROM	12x	52x і вище
Операційна система	Microsoft Windows Server 2016	
Сервер баз даних	Microsoft SQL Server 2017	
Канал Інтернета, корпоративної або локальної мережі	5 Мбіт/с	10 Мбіт/с і вище

Програмні вимоги для сервера

Таблиця 3

Ресурс	**Мінімальний**	**Рекомендований**
Процесор	2 ГГц	2.5 ГГц і вище
Оперативна пам’ять	2 Гб	4 Гб і вище
Жорсткий диск	40 Гб	100 Гб і вище
Канал Інтернета, корпоративної або локальної мережі	20 Мбіт/с	100 Мбіт/с і вище
Роздільність монітору	1280x800	
Привод CD-ROM	4x	40x і вище
Відеокарта	1 Гб	2 Гб та вище

Ще одним важливим компонентом для функціонування системи є програмне забезпечення. Для коректної роботи та підтримки узгодженої роботи розробленої облікової системи було обрано таке серверне програмне забезпечення, необхідне для забезпечення оптимальної продуктивності системи.

На сервері використовується **Microsoft Server 2010** або новіша версія, а також **SQL Server 2012** або новіша версія.

Операційна система, встановлена на клієнтських комп’ютерах, — **Microsoft Windows 7** (або 8 чи 10).

Для формування звітів використовується програма **Microsoft Word 2010** або новіша версія.

4.3 Опис програми

Після запуску роботи з інформаційною обліковою системою для магазину будівельних матеріалів користувачу відображається **вікно авторизації**, показане на Рисунках 9 та 15. У цьому вікні користувач повинен ввести **логін** і **пароль.**

Логін і пароль зазвичай різняться залежно від ролей: **менеджер**, **бухгалтер**, **комірник** та **логіст**.

Подальша робота з інформаційною системою проілюстрована на Рисунках з 4.2 по 4.16.

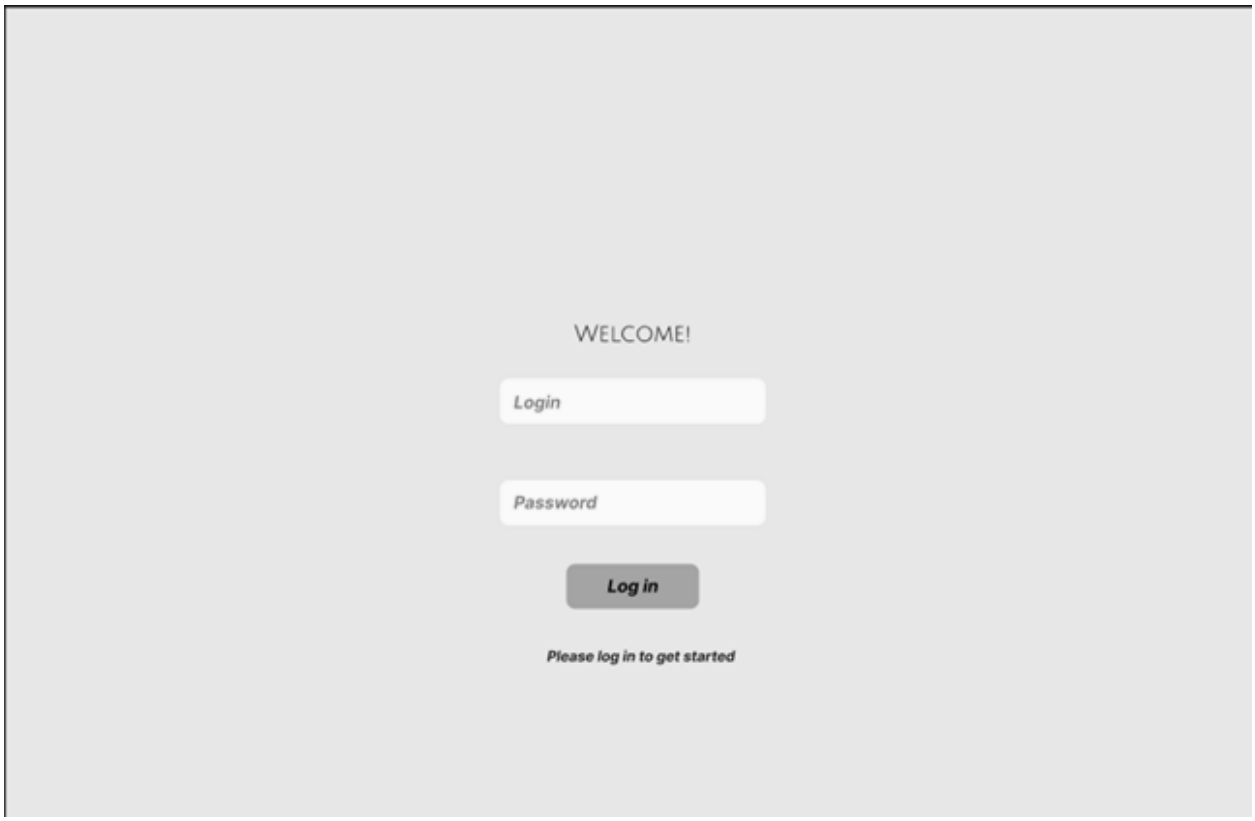


Рис. 4.2 Форма авторизації для всіх

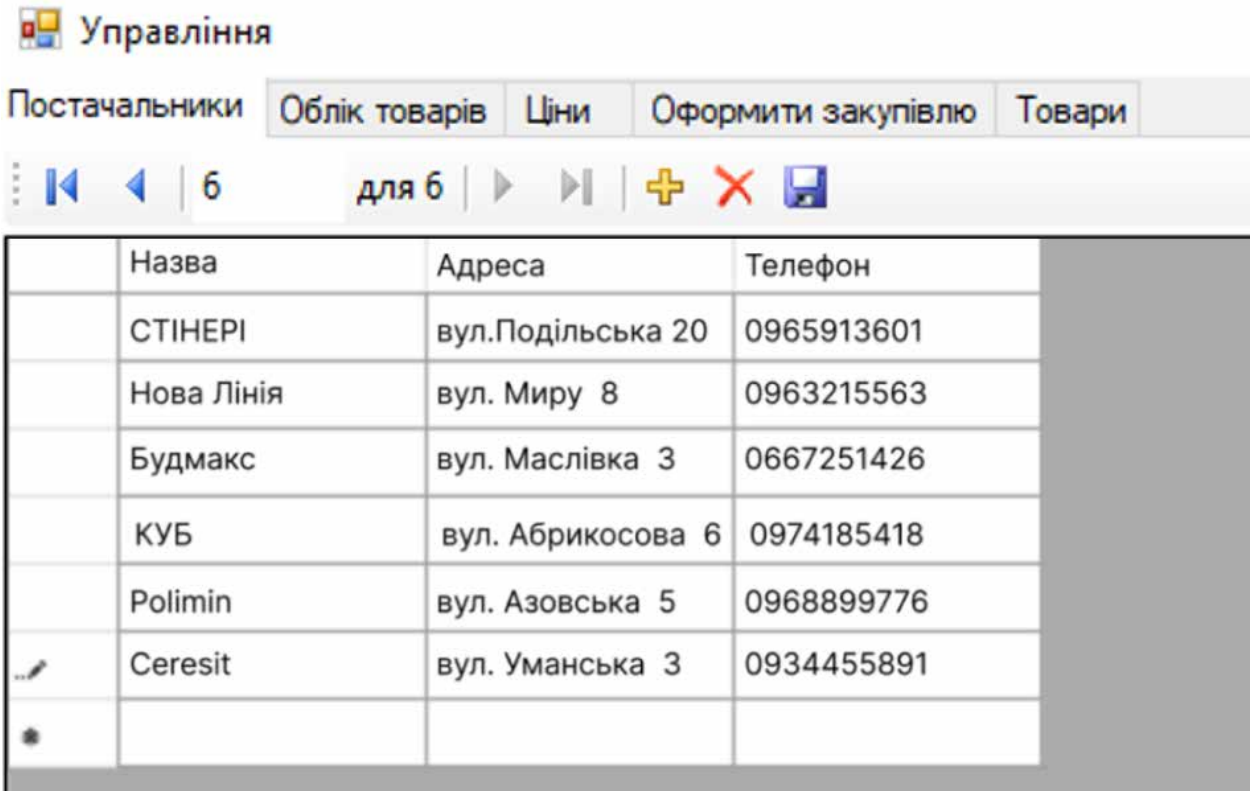


Рис. 4.3 Вікно введення постачальників

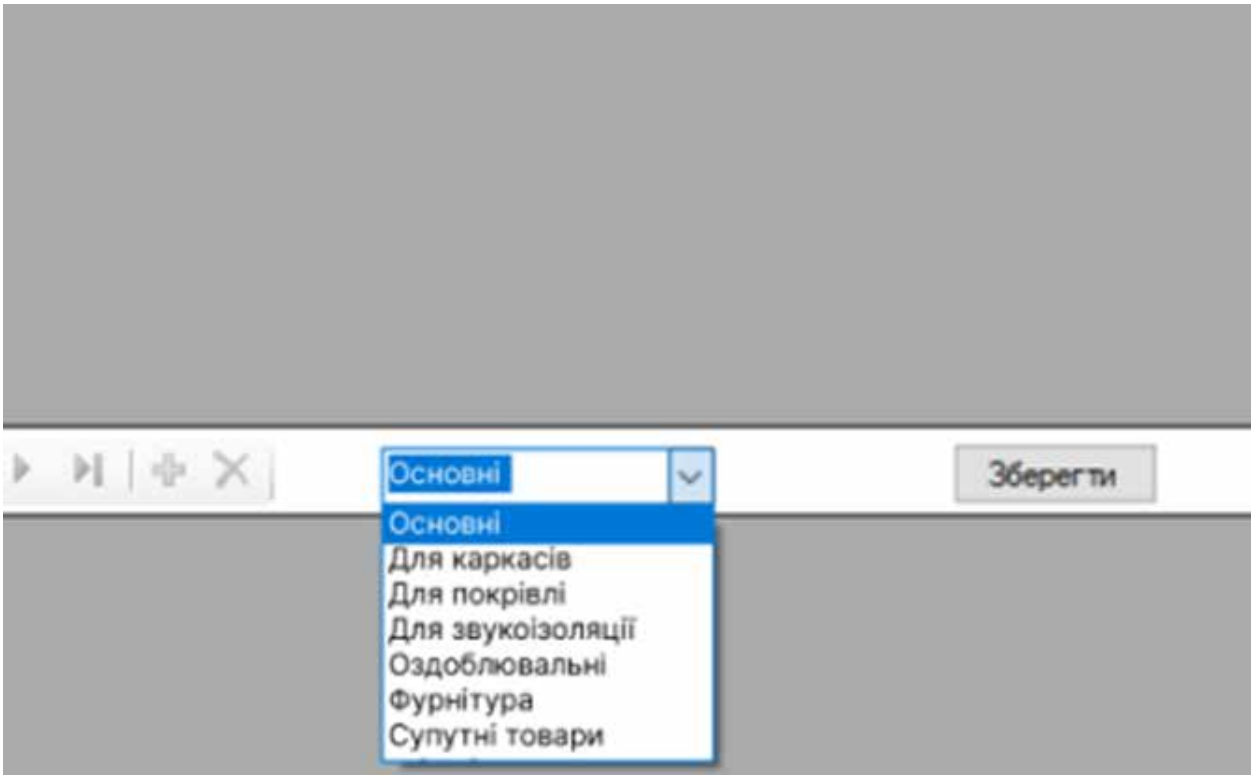


Рис. 4.4 Вибір виду Будівельних матеріалів

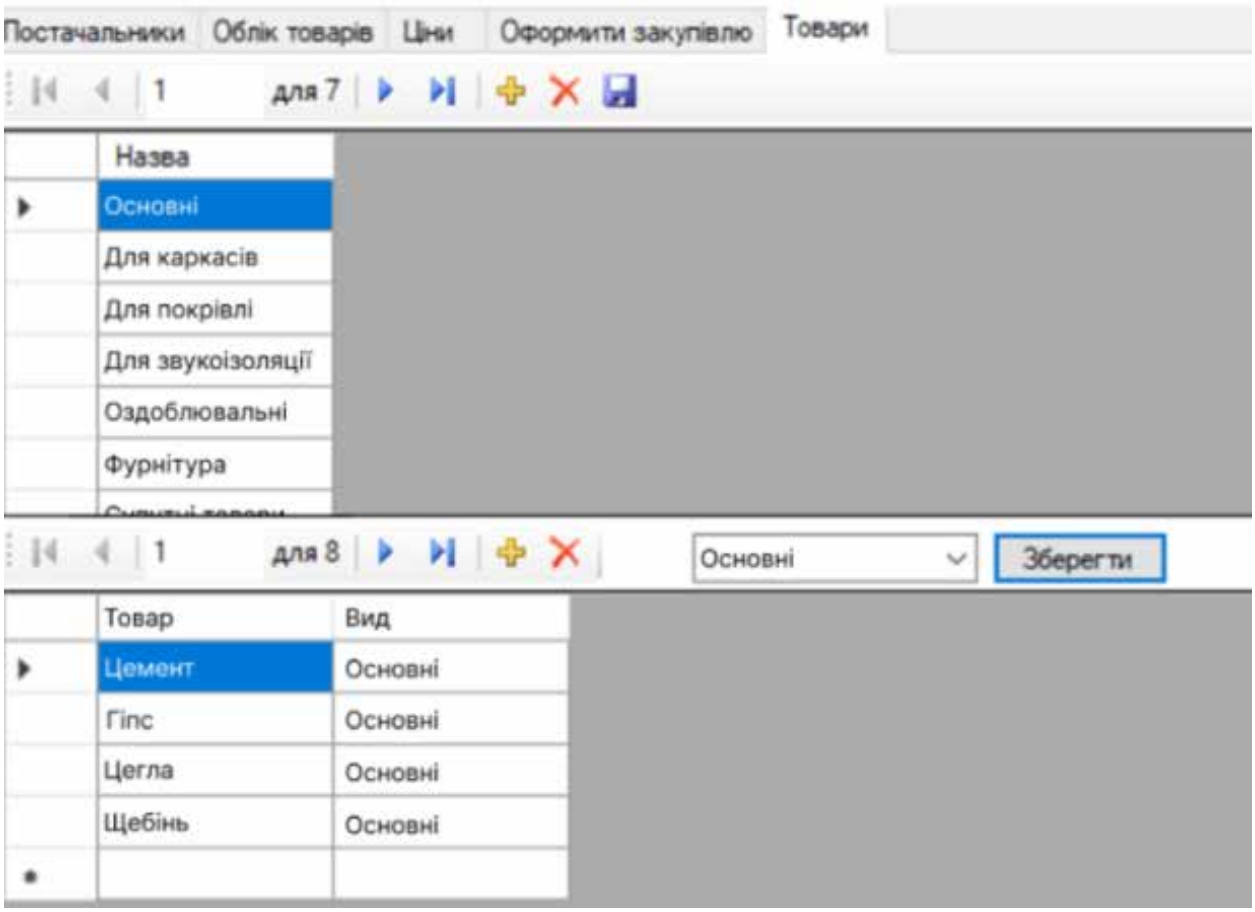


Рис. 4.5 Введення товар

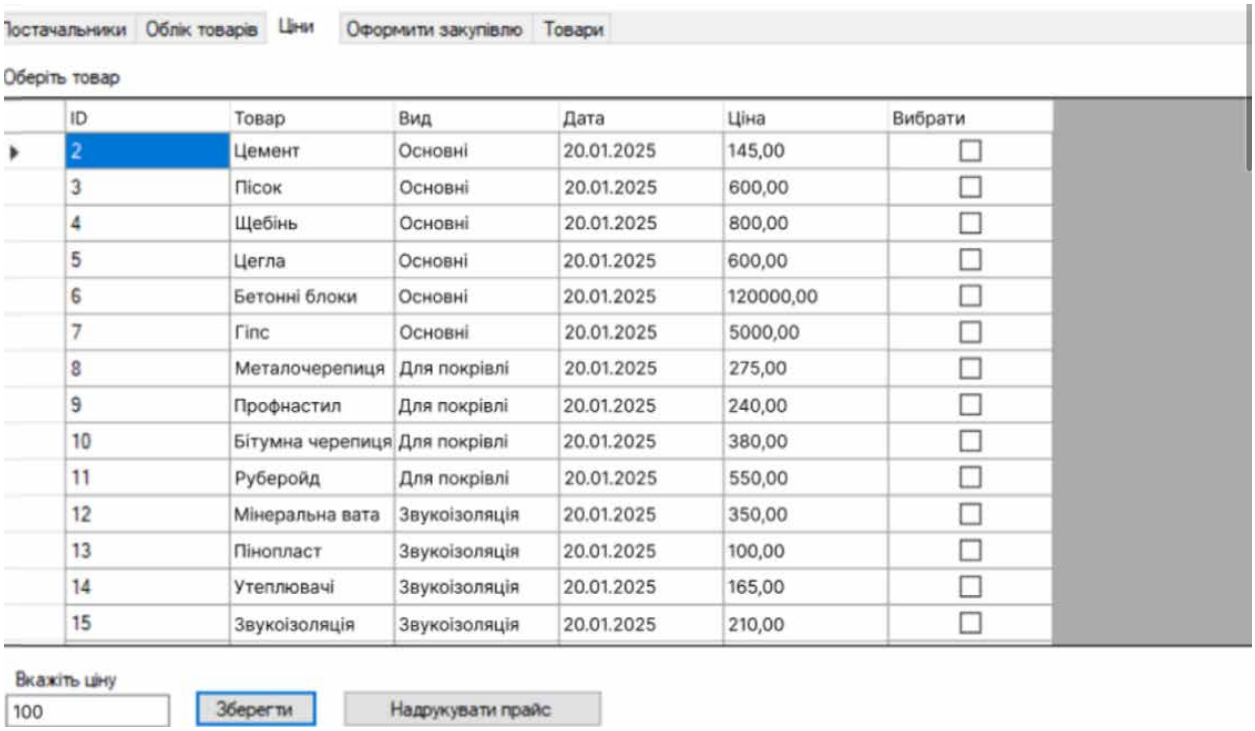


Рис. 4.6 Вікно керування прайс-листом

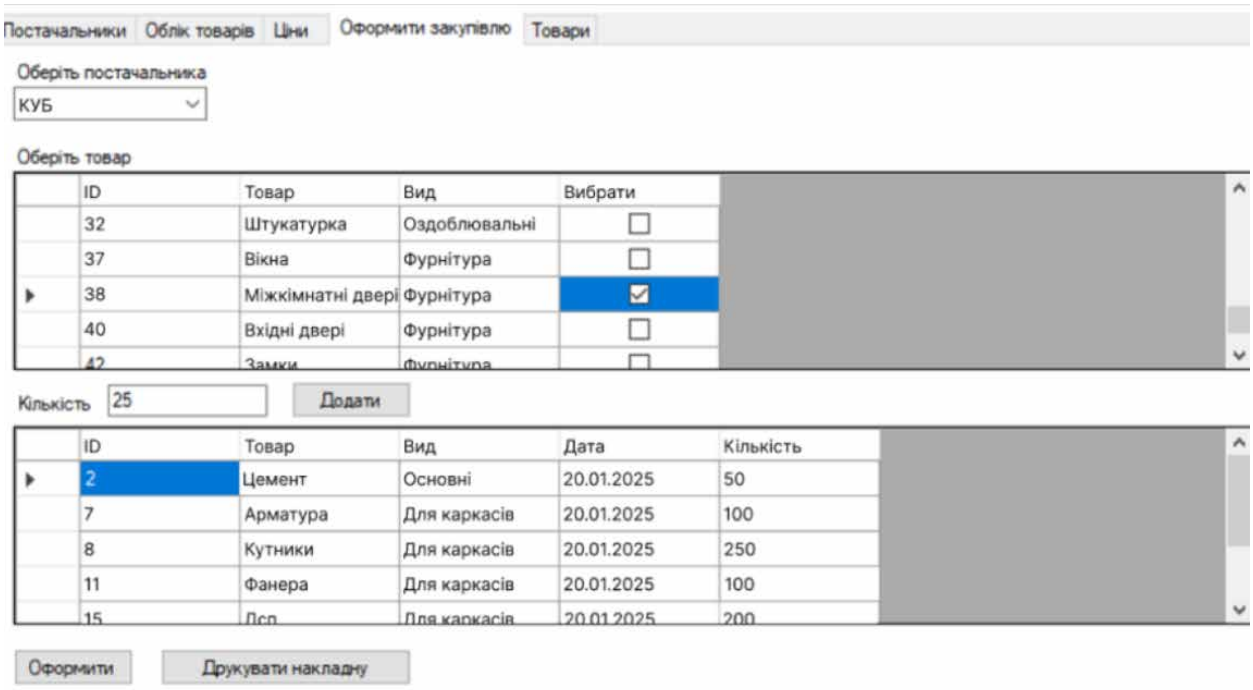


Рис. 4.7 Вибір необхідних до закупівлі товарів та оформлення закупівлі

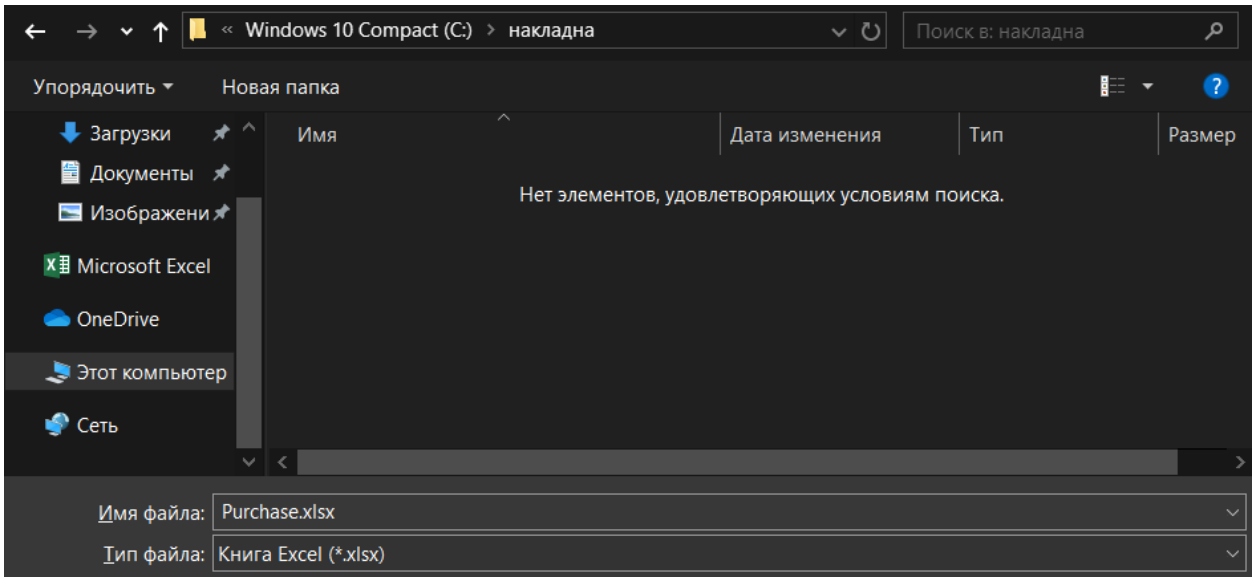
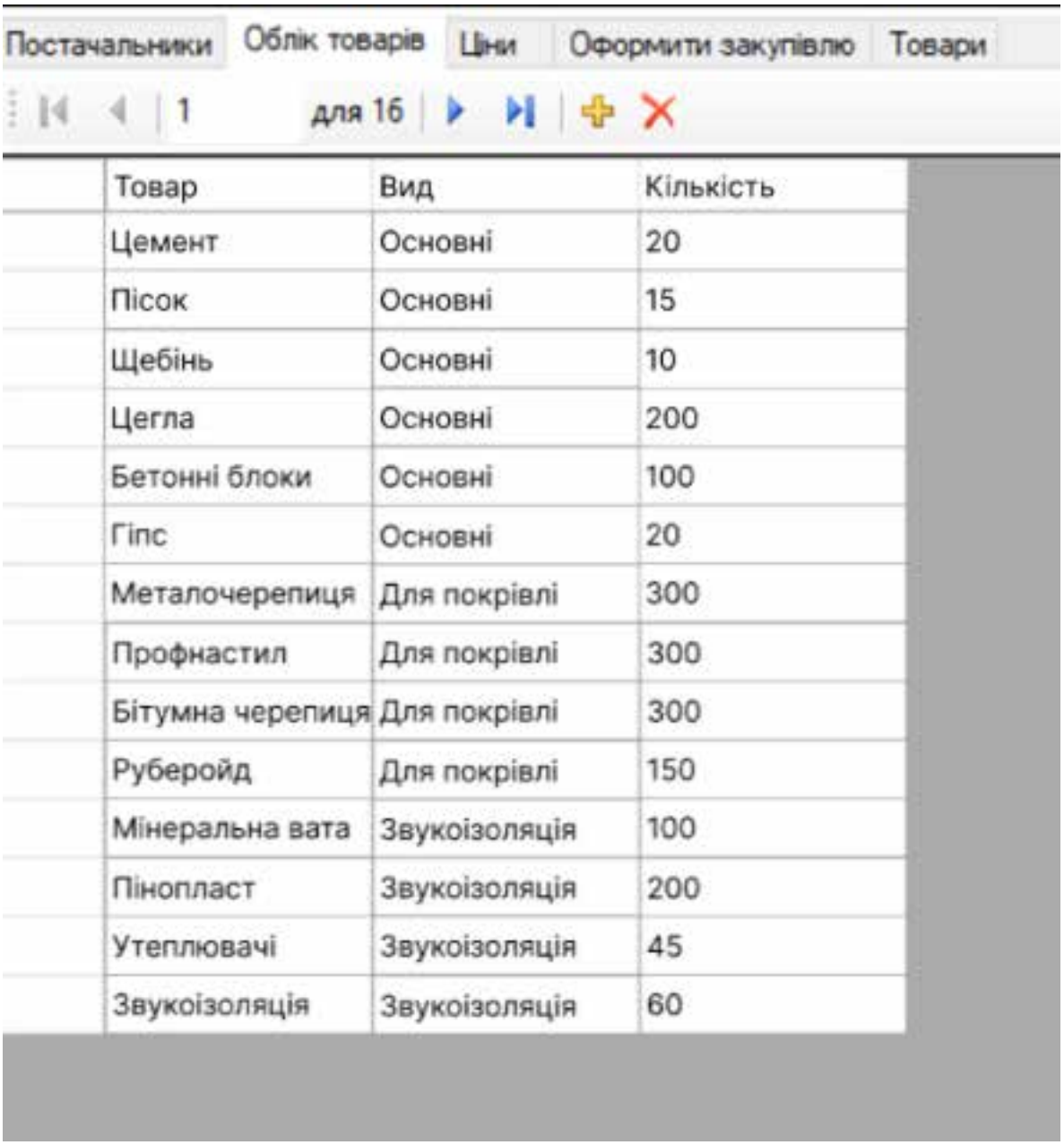


Рис. 4.8 Збереження накладної закупівлі

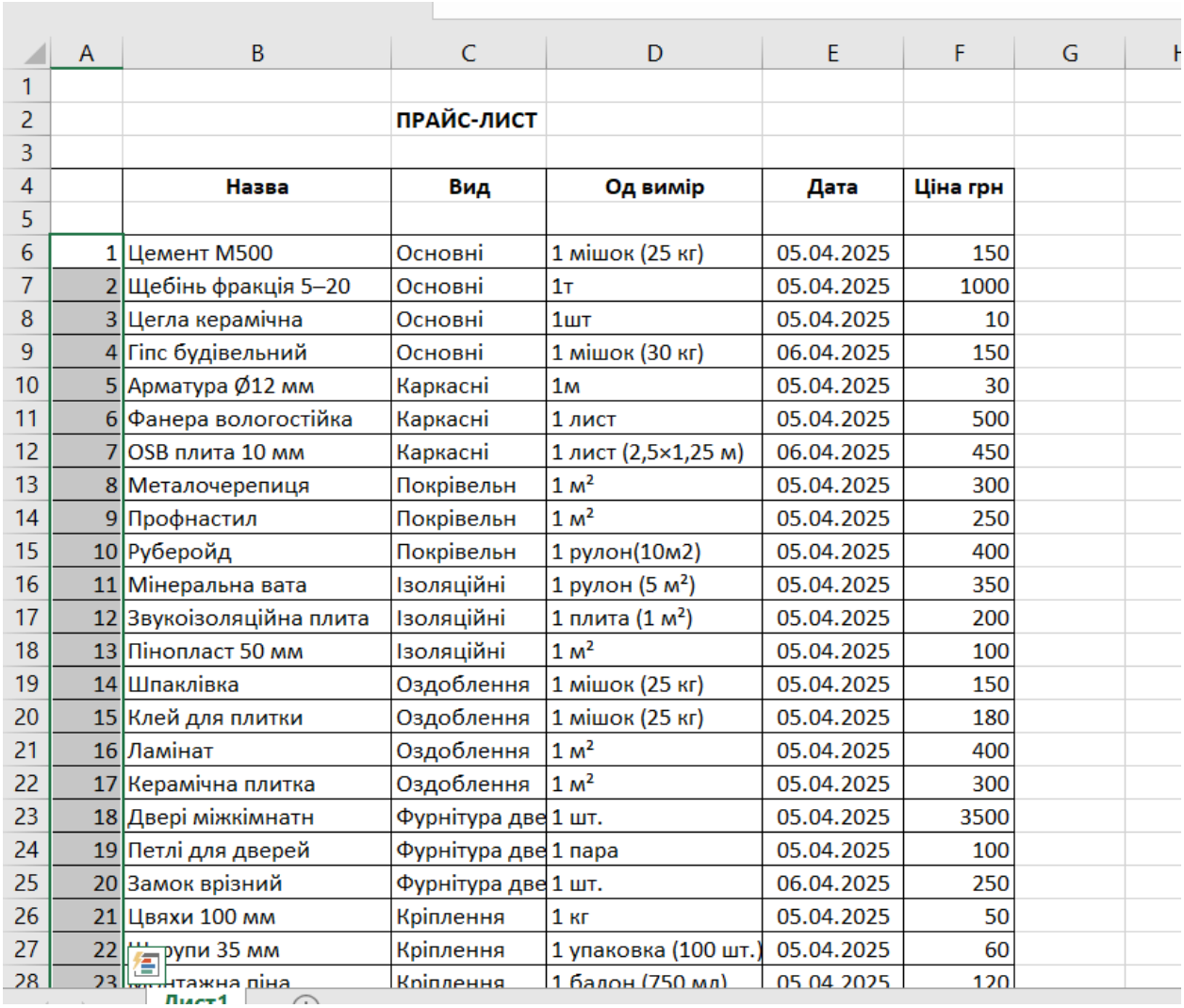
	A	B	C	D	E	F	G	H
1								
2		Постачальник	КУБ					
3			вул.Абрикосова 6					
4								
5								
6								
7		Одержувач		materials				
8								
9								
10								
11			**Рахунок-фактура №3**					
12			від 21.03.2025					
13								
14								
15								
16								
17	**№**	**Назва**	**вид**	**од. вимір**	**кількість**	**ціна**	**сума**	
18	1	Цегла	Основні	шт	200	10	2000	
19	2	Арматура	Для каркасів	1м	5	30	150	
20	3	Металочерепиця	Для покрівлі	1м2	10	290	2900	
21	4	Пінопласт 50мм	Звукоізоляція	1м2	10	100	1000	
22	5	Двері міжкімнатні	Фурнітура	шт	1	3500	3500	
23								
24					**всього**		**9550**	
25								
26					**виписав**			
27								

Рис. 4.9 Накладна закупівлі



	Товар	Вид	Кількість
	Цемент	Основні	20
	Пісок	Основні	15
	Щебінь	Основні	10
	Цегла	Основні	200
	Бетонні блоки	Основні	100
	Гіпс	Основні	20
	Металочерепиця	Для покрівлі	300
	Профнастил	Для покрівлі	300
	Бітумна черепиця	Для покрівлі	300
	Руберойд	Для покрівлі	150
	Мінеральна вата	Звукоізоляція	100
	Пінопласт	Звукоізоляція	200
	Утеплювачі	Звукоізоляція	45
	Звукоізоляція	Звукоізоляція	60

Рис. 4.10 Вікно перегляду обліку товарів



	A	B	C	D	E	F	G	H
1								
2			**ПРАЙС-ЛИСТ**					
3								
4		**Назва**	**Вид**	**Од вимір**	**Дата**	**Ціна грн**		
5								
6	1	Цемент М500	Основні	1 мішок (25 кг)	05.04.2025	150		
7	2	Щебінь фракція 5–20	Основні	1т	05.04.2025	1000		
8	3	Цегла керамічна	Основні	1шт	05.04.2025	10		
9	4	Гіпс будівельний	Основні	1 мішок (30 кг)	06.04.2025	150		
10	5	Арматура Ø12 мм	Каркасні	1м	05.04.2025	30		
11	6	Фанера вологостійка	Каркасні	1 лист	05.04.2025	500		
12	7	OSB плита 10 мм	Каркасні	1 лист (2,5×1,25 м)	06.04.2025	450		
13	8	Металочерепиця	Покрівельн	1 м²	05.04.2025	300		
14	9	Профнастил	Покрівельн	1 м²	05.04.2025	250		
15	10	Руберойд	Покрівельн	1 рулон(10м2)	05.04.2025	400		
16	11	Мінеральна вата	Ізоляційні	1 рулон (5 м²)	05.04.2025	350		
17	12	Звукоізоляційна плита	Ізоляційні	1 плита (1 м²)	05.04.2025	200		
18	13	Пінопласт 50 мм	Ізоляційні	1 м²	05.04.2025	100		
19	14	Шпаклівка	Оздоблення	1 мішок (25 кг)	05.04.2025	150		
20	15	Клей для плитки	Оздоблення	1 мішок (25 кг)	05.04.2025	180		
21	16	Ламінат	Оздоблення	1 м²	05.04.2025	400		
22	17	Керамічна плитка	Оздоблення	1 м²	05.04.2025	300		
23	18	Двері міжкімнатн	Фурнітура две	1 шт.	05.04.2025	3500		
24	19	Петлі для дверей	Фурнітура две	1 пара	05.04.2025	100		
25	20	Замок врізний	Фурнітура две	1 шт.	06.04.2025	250		
26	21	Цвяхи 100 мм	Кріплення	1 кг	05.04.2025	50		
27	22	Шурупи 35 мм	Кріплення	1 упаковка (100 шт.)	05.04.2025	60		
28	23	Монтажна піна	Кріплення	1 балон (750 мл)	05.04.2025	120		

Рис. 4.11 Файл прайс-листа до друку

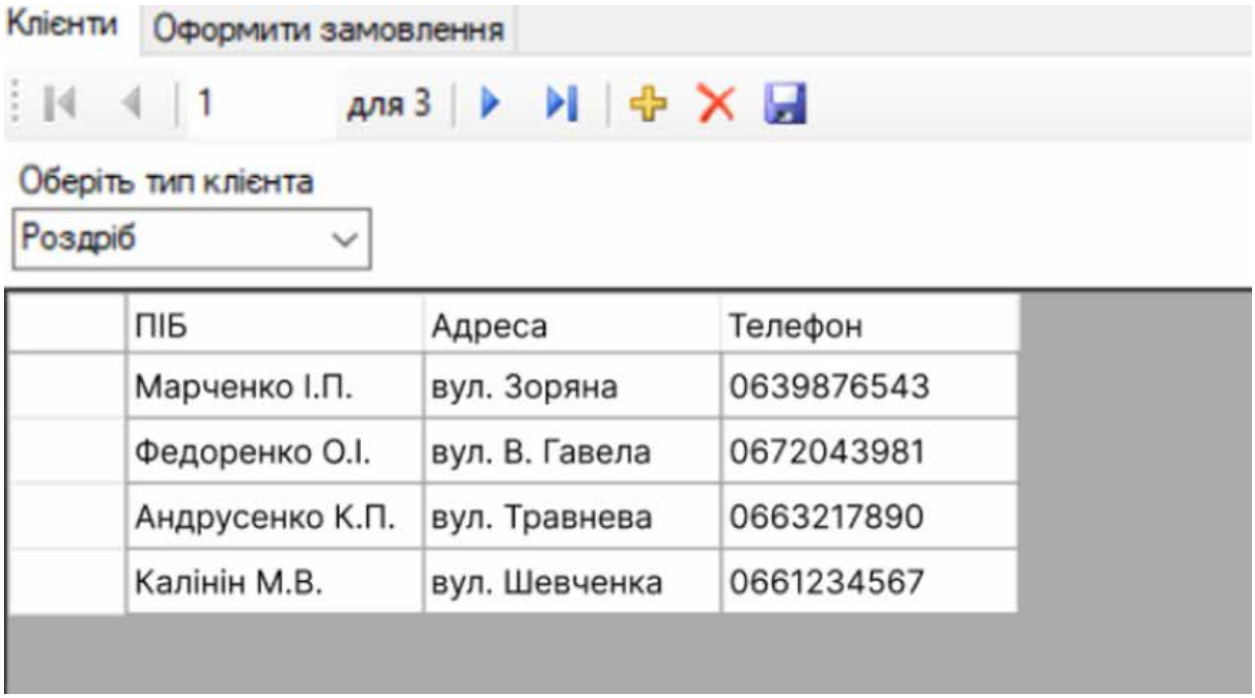


Рис 4.12 Продавець , який вводе роздрібного клієнта

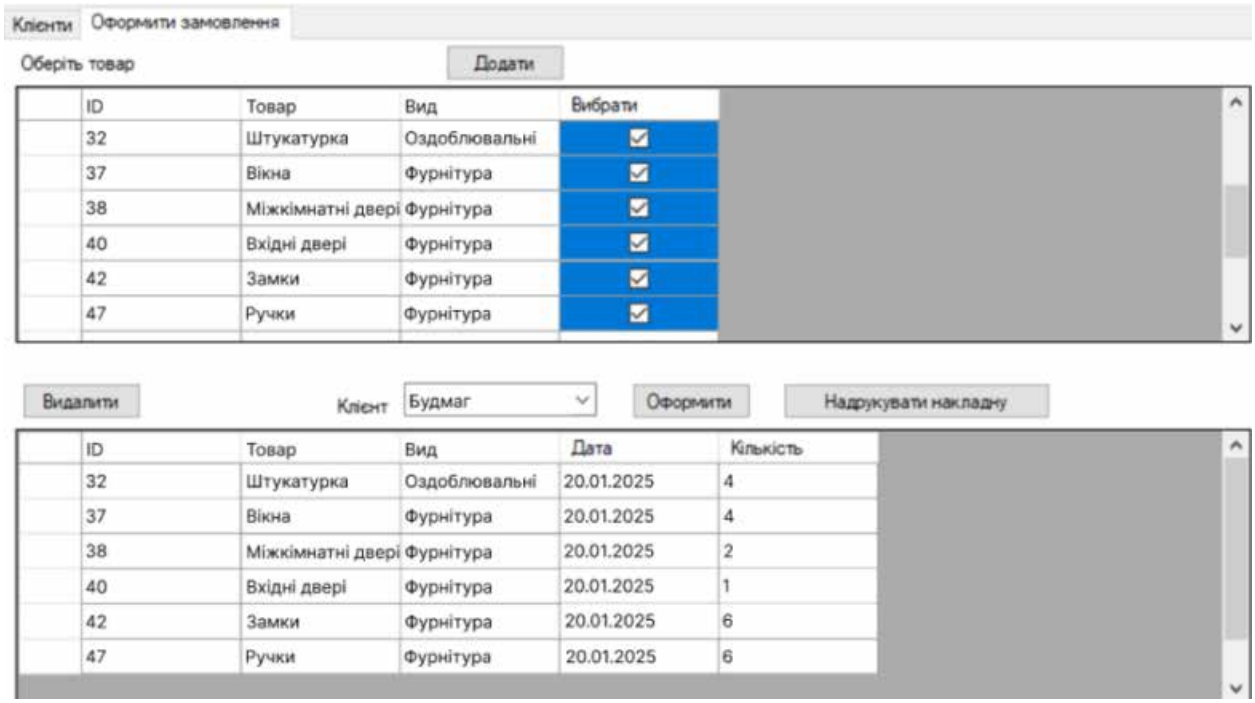


Рис. 4.13 Оформлення замовлення

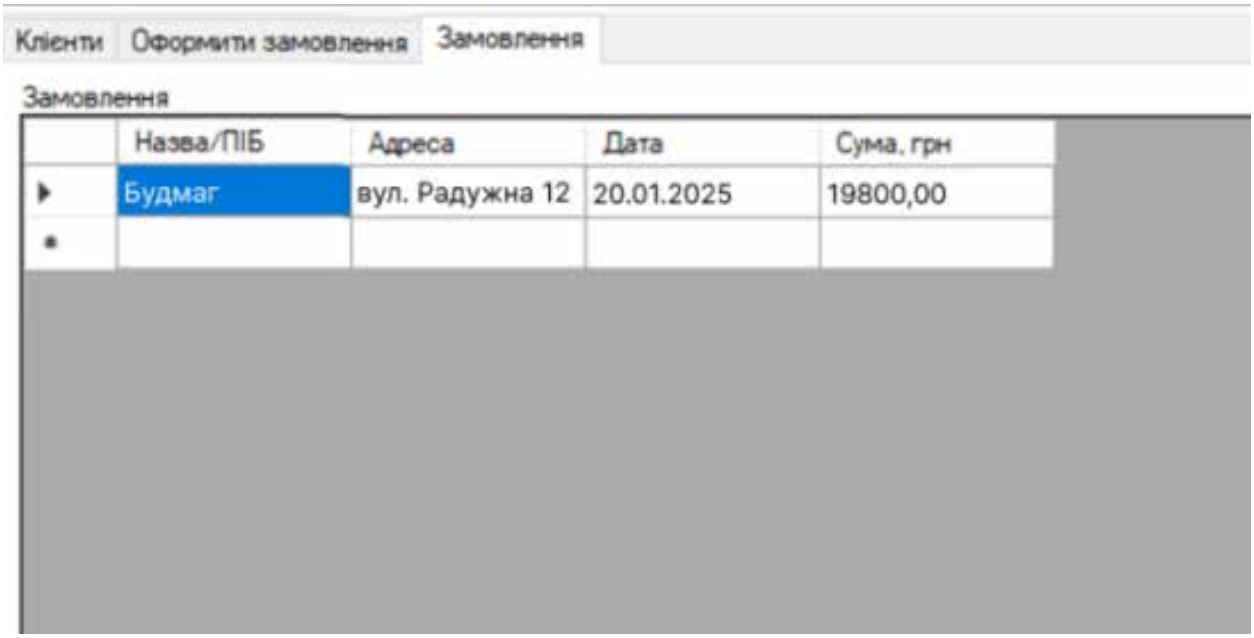


Рис. 4.14 Вікно перегляду проведених замовлень

Товарний чек

Назва	Вид	Дата	Кількість	Ціна	Сума
Цемент М500	Основні	20.01.2025	3	150	450
Цегла керамічна	Основні	20.01.2025	100	10	1000
Гіпс будівельний	Основні	20.01.2025	2	150	300
Арматура Ø12 мм	Каркасні	20.01.2025	9	30	270
Фанера вологостійка	Каркасні	20.01.2025	3	500	1500
OSB плита 10 мм	Каркасні	20.01.2025	3	450	1350
			Всього:		**4870**

Рис. 4.15 Товарний чек замовлення

Облік товарів

1 для 16

	Товар	Вид	Кількість
	Цемент	Основні	20
	Пісок	Основні	15
	Щебінь	Основні	10
	Цегла	Основні	200
	Бетонні блоки	Основні	100
	Гіпс	Основні	20
	Металочерепиця	Для покрівлі	300
	Профнастил	Для покрівлі	300
	Бітумна черепиця	Для покрівлі	300
	Руберойд	Для покрівлі	150
	Мінеральна вата	Звукоізоляція	100
	Пінопласт	Звукоізоляція	200
	Утеплювачі	Звукоізоляція	45
	Звукоізоляція	Звукоізоляція	60

Рис. 4.16 Комірник, який переглядає облік товарів

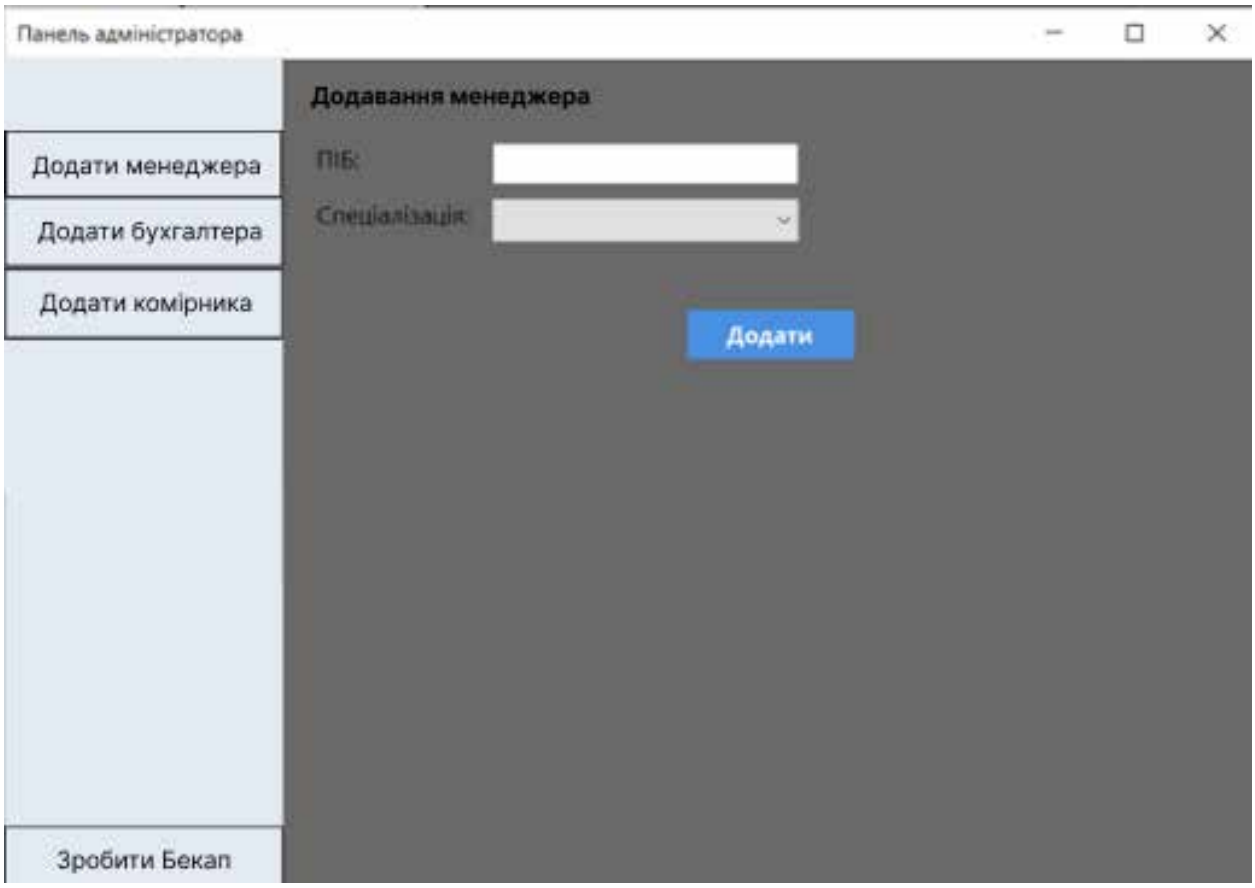


Рис. 4.17 Додавання менеджера

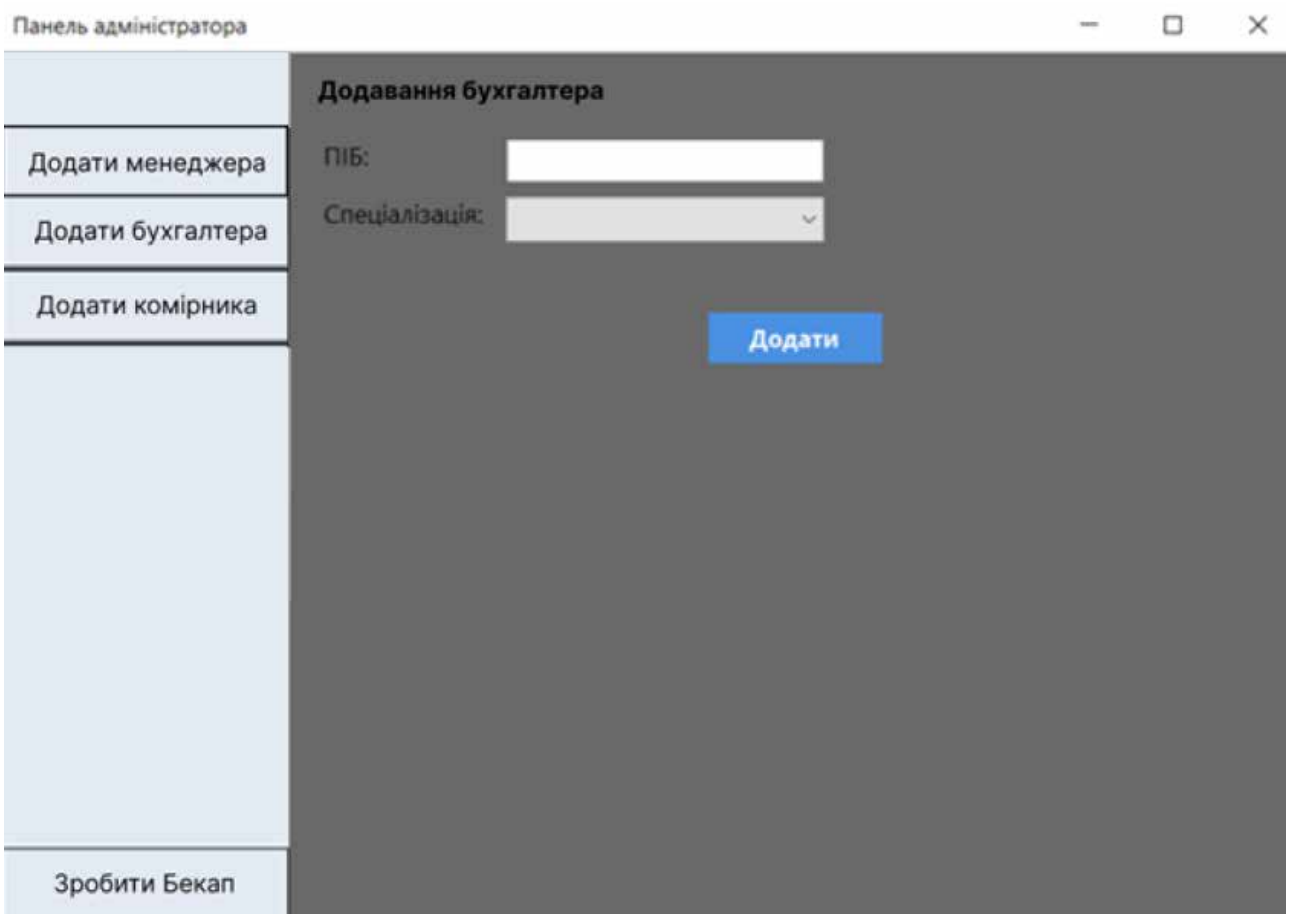


Рис. 4.18 Додавання бухгалтера

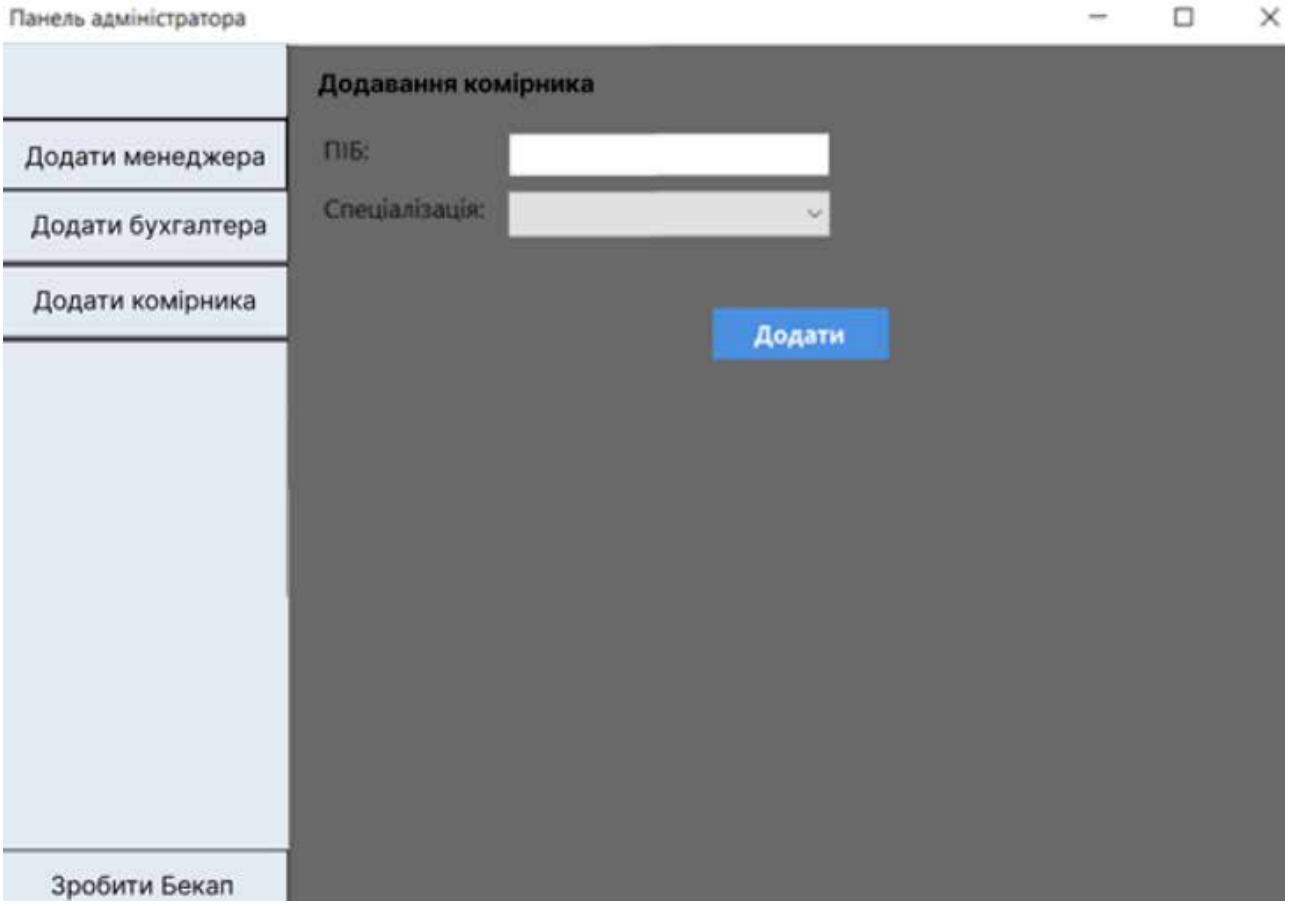


Рис. 4.19 Додавання комірника

ВИСНОВОК

У результаті виконання цього дипломного проєкту було розроблено інформаційну систему управління для контролю будівельних матеріалів. Система була спроєктована відповідно до встановлених цілей. Була розроблена проста, зрозуміла та зручна для користувача база даних.

Це рішення дозволяє працівнику магазину будівельних матеріалів використовувати застосунок у щоденній роботі, автоматизувати процес управління замовленнями та обліком на складі, а також створити базу даних клієнтів.

Для реалізації цього завдання було обрано такі програмні засоби:

Під час розробки проєкту використовувалося середовище розробки **MS SQL Server**, яке базується на реляційних технологіях.

У ході проєкту були враховані три обов’язкові рівні взаємодії клієнт–сервер:

- **рівень представлення**, що відповідає за введення даних та їх відображення користувачеві;
- **прикладний рівень**, який реалізує бізнес-логіку та обробляє всю необхідну інформацію;
- **рівень управління даними**, який забезпечує зберігання даних та доступ до них.

Як середовище розробки застосунку було обрано **Visual Studio** — систему, що використовується для створення програмних продуктів мовою програмування **C#**. Visual Studio є зручним і практичним середовищем. Яскравим прикладом є використання методу **перетягування (drag-and-drop)**, що робить візуальну розробку достатньо інтуїтивно зрозумілою. Це програмне забезпечення має низку доречних переваг, зокрема:

- створення запитів безпосередньо в програмі;

- робота з базами даних;
- зручний інтерфейс та оформлення.

У **першому розділі** проєкту були визначені основні завдання, встановлені вхідні та вихідні дані, а також проаналізовано процес замовлення товарів. **Вхідні дані** містять інформацію про постачальників, клієнтів та замовлену продукцію. **Вихідні дані** включають касовий чек або рахунок-фактуру, звіт про продажі та поточний залишок товару на складі.

Було проведено **моделювання системи** та **системний аналіз**. Під час моделювання були створені діаграми з використанням мови моделювання **UML**.

У **другому розділі** було обрано інформаційне забезпечення, а саме мову **SQL**, **SQL Server** та середовище **MS SQL Server**.

У **третьому розділі** для розробки прикладного програмного забезпечення було обрано мову програмування **C#** та середовище **Visual Studio**.

У **останньому розділі** були розглянуті види тестування програмного забезпечення. Визначено необхідні вимоги до апаратного та програмного забезпечення. Було описано функціонування системи, проведено тестовий продаж і сформовано **прайс-лист** та **рахунок-фактуру**.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ [Електронний ресурс] – Режим доступу до ресурсу: https://studfiles.net/preview/5828440/.
2. Системы управління базами данних [Електронний ресурс] – Режим доступу до ресурсу: http://wm-help.net/lib/b/book/120467185/94.
3. Основы языка SQL [Електронний ресурс] – Режим доступу до ресурсу: http://www.ict.edu.ru/ft/004864/sql.pdf.
4. Бовуар, А. SQL. Основи, написання запитів і проєктування баз даних / А. Бовуар. — Харків : Видавництво «Фабула», 2020. — 320 с.
5. Бофілос, І. SQL для початківців / І. Бофілос. — Київ : Літера ЛТД, 2021. — 240 с.
6. Больдт, Т. SQL за 10 хвилин / Т. Больдт ; пер. з англ. — Львів : Видавництво Старого Лева, 2020. — 256 с.
7. Болюк, С. В. Бази даних та SQL : навч. посібник / С. В. Болюк, А. М. Мазур. — Київ : НАУ, 2019. — 212 с.
8. Язык программирования C++ [Електронний ресурс] – Режим доступу до ресурсу: https://goo.gl/8aHfWM.
9. Страуструп, Б. Програмування. Принципи та практика з використанням C++ / Б. Страуструп ; пер. з англ. — Київ : Діалектика, 2020. — 1024 с.
10. Ліпман, С. Б., Лажой, Ж. C++ для професіоналів / С. Б. Ліпман, Ж. Лажой. — Київ : Видавнича група BHV, 2021. — 640 с.
11. Еккель, Б. Філософія C++ / Б. Еккель ; пер. з англ. — Харків : Видавництво «Фабула», 2019. — 768 с.

12. Шилдт, Г. C++ : повний курс / Г. Шилдт. — Київ : Наука і техніка, 2018. — 960 с.
13. Ярославський, М. М. Об'єктно-орієнтоване програмування мовою C++ : навч. посібник / М. М. Ярославський. — Київ : КНЕУ, 2020. — 210 с.
14. Карвин Б. Программирование баз данных SQL. Типичные ошибки и их устранение / Билл Карвин., 2011. – 336 с.
15. Буч Г., Якобсон А., Рамбо Дж. UML. Классика CS / С. Орлов. — 2-е изд.. — СПб.: Питер, 2006. — 736 с.
16. Unified Modeling Language (UML) Resource Page [Електронний ресурс] – Режим доступу до ресурсу: http://www.uml.org/.