Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної (циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: «Розроблення клієнт-серверної системи управління замовленнями та складським обліком для кондитерського підприємства»

Виконав студент 2 курсу, групи КНС-21
спеціальності:
122 „Комп'ютерні науки"
(шифр і назва напрямку підготовки спеціальності)

Кіндрат Олег Юрійович
(прізвище, ім'я, по батькові)

Керівники: Лизанчук Т.С.
(прізвище, ініціали)

Думанський О.І.
(прізвище, ініціали)

Рецензент: Гонцар Р.О.
(прізвище, ініціали)

Львів-2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 "Комп'ютерні науки"

ЗАТВЕРДЖУЮ:
Завідувачка кафедри КН
_____ Борецька І.Б.
„10" червня 2025 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Кіндрат Олег Юрійович
(прізвище, ім'я, по батькові)

1.Тема бакалаврської роботи: "Розроблення клієнт-серверної системи управління замовленнями та складським обліком для кондитерського підприємства", затверджені наказом вищого навчального закладу від "15" листопада 2024 року, №С-882.
2.Термін подання студентом проекту (роботи) 10 червня 2025 р.
3. Вихідні дані до проекту (роботи) Створити клієнт-серверну систему управління замовленнями та складським обліком для кондитерського підприємства. Для розроблення програмного забезпечення використати мову програмування C#.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Стан проблемної області
Інформаційне та математичне забезпечення
Програмне та технічне забезпечення
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до доповіді.
6. Дата видачі завдання 18 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№. з/п	Етапи бакалаврської роботи	Терміи виконання етапів роботи	Примітка
1.	Аналіз предметної області	18.11.24 – 09.12.24	виконано
2.	Вибір засобів розроблення	10.12.24 – 20.12.24	виконано
3.	Розроблення та наповнення БД	21.12.24 – 22.01.25	виконано
4.	Розроблення структури програми	23.01.25 – 17.02.25	виконано
5.	Програмна реалізація	18.02.25 – 16.04.25	виконано
6.	Тестування роботи програми	17.04.25 – 10.05.25	виконано
7.	Оформлення звіту до диплому	11.05.25 – 10.06.25	виконано

Студент _____ Кіндрат О.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Лизанчук Т.С.
(підпис) (прізвище та ініціали)

_____ Думанський О.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 51 сторінок пояснювальної записки, 11 рисунків, 2 додатка, 15 джерел.

У дипломній роботі розглянуто процес розроблення інформаційної системи для автоматизації управління складом та обробки замовлень у кондитерському бізнесі. Система призначена для ведення обліку запасів сировини та готової продукції, контролю замовлень клієнтів, управління ролями користувачів (адміністратор, менеджер, комірник) та забезпечує інтеграцію з базою даних MySQL. Впроваджені функції включають реєстрацію, автентифікацію, контроль залишків на складі, обробку замовлень та ведення історії операцій.

Ключові слова: інформаційна система, управління складом, замовлення, база даних, кондитерський бізнес, MySQL, C#, автоматизація, облік запасів, рольова модель.

ABSTRACT

The thesis consists of 51 pages of explanatory text, 11 figures, 2 appendices, and 15 sources.

The diploma thesis discusses the development process of an information system for automating warehouse management and order processing in the confectionery business. The system is designed to maintain inventory records of raw materials and finished products, manage customer orders, handle user roles (administrator, manager, warehouse clerk), and provide integration with a MySQL database. Implemented features include user registration, authentication, stock control, order processing, and operation history tracking.

Keywords: information system, warehouse management, orders, database, confectionery business, MySQL, C#, automation, inventory accounting, role-based model.

ТЕХНІЧНЕ ЗАВДАННЯ

У межах дипломної роботи передбачається розроблення клієнт-серверної інформаційної системи для автоматизації процесів управління замовленнями та обліком товарів на складі кондитерського підприємства. Передбачено створення системи, яка дозволяє ефективно вести облік готової продукції, сировини, клієнтів, постачальників та оперативно обробляти замовлення. Система повинна бути зручною для використання персоналом підприємства та сприяти зменшенню часу на виконання щоденних завдань.

Система повинна базуватися на клієнт-серверній архітектурі з використанням спільної централізованої бази даних. Клієнтська частина повинна бути реалізована у вигляді десктопного застосунку мовою програмування C# із застосуванням бібліотек Windows Forms або WPF, тоді як серверна частина – у вигляді бази даних MySQL з доступом через мережу.

Основна функціональність системи повинна включати: облік готової продукції, визначення її складу з інгредієнтів, ведення довідників постачальників, клієнтів, матеріалів; створення, редагування та виконання замовлень із автоматичним списанням відповідної продукції та сировини зі складу. Також необхідно передбачити автоматичне оновлення залишків, контроль мінімальних запасів, формування звітності за періодами: за замовленнями, витратами, обсягами продажів, залишками тощо.

Користувачі системи повинні проходити авторизацію з розмежуванням прав доступу відповідно до ролей, зокрема адміністратор, менеджер чи комірник. Для забезпечення безпеки необхідно передбачити шифрування паролів, а також ведення журналу дій користувачів. Інтерфейс програми має бути багатовіконним, інтуїтивно зрозумілим, з підтримкою стандартних функцій пошуку та фільтрації по таблицях і попереджень про критичні залишки.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ІС	– Інформаційна система
СКБД	– Система керування базами даних
SQL	– Structured Query Language – мова структурованих запитів
БД	– База даних
PK	– Primary Key – первинний ключ
FK	– Foreign Key – зовнішній ключ
UI	– User Interface – інтерфейс користувача
MySQL	– СКБД, що використовується у проєкті
C#	– Мова програмування C Sharp
ER-діаграма	– Entity-Relationship діаграма – діаграма сутність-зв'язок
Лог	– Журнал подій – запис дій системи
UML	– Unified Modeling Language – уніфікована мова моделювання
Hash	– Хешування пароля для збереження безпеки

ВСТУП

Сучасні умови господарювання вимагають від підприємств ефективного управління ресурсами, оперативного прийняття рішень та оптимізації виробничих процесів. Зокрема, для малих і середніх виробників у сфері харчової промисловості, зокрема кондитерських підприємств, важливою є автоматизація таких процесів, як управління замовленнями, облік залишків сировини й готової продукції, моніторинг поставок та взаємодія з клієнтами. Впровадження інформаційних технологій у ці процеси дозволяє підвищити ефективність виробництва, мінімізувати втрати та підвищити рівень обслуговування споживачів.

У зв’язку з цим, одним із перспективних напрямів цифрової трансформації є створення клієнт-серверних інформаційних систем, що забезпечують централізоване управління даними при збереженні локального доступу для персоналу підприємства. Такі системи дозволяють об’єднати в одному рішенні облік ресурсів, оперативне формування та виконання замовлень, контроль залишків та аналіз фінансової та складської інформації.

Актуальність теми полягає у необхідності розроблення доступного та зручного інструменту для автоматизації облікових і виробничо-комерційних процесів на прикладі умовного кондитерського підприємства.

Об'єкт дослідження – процеси управління замовленнями та складським обліком на малому підприємстві у сфері харчової промисловості.

Предмет дослідження – програмні засоби, алгоритми та моделі, що забезпечують функціонування клієнт-серверної інформаційної системи обліку замовлень і ресурсів підприємства.

Метою дипломної роботи є розроблення клієнт-серверної інформаційної системи для автоматизації управління замовленнями та обліком складських ресурсів на прикладі кондитерського підприємства.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

- провести аналіз предметної області та сучасних підходів до автоматизації облікових процесів;

- визначити вимоги до інформаційної системи, її функціональність та архітектуру;

- розробити інформаційну та математичну моделі основних об'єктів системи;

- спроєктувати структуру бази даних із використанням СКБД MySQL;

- реалізувати клієнтську частину системи мовою програмування C# з підтримкою графічного інтерфейсу;

- забезпечити механізми авторизації, розмежування прав доступу, шифрування та логування дій користувачів;

- протестувати функціональність системи, перевірити її працездатність, продуктивність та зручність використання.

Методи дослідження, що використовуються у дипломній роботі:

- методи аналізу та узагальнення – для вивчення предметної області;

- структурне моделювання – для побудови інформаційної та логічної архітектури системи;

- об’єктно-орієнтоване проєктування – при створенні програмного коду;

- методи модульного, інтеграційного та функціонального тестування – для перевірки коректності роботи системи;

- засоби реляційного моделювання – для проєктування структури бази даних.

У результаті виконання роботи планується отримання повнофункціонального десктопного застосунку, здатного ефективно підтримувати ключові облікові й адміністративні процеси невеликого виробничого підприємства. Реалізоване рішення може бути адаптоване під специфіку інших бізнесів або розширене в майбутньому за рахунок інтеграції з мобільними чи веб-платформами.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Аналіз предметної області

Сфера виробництва та реалізації кондитерської продукції в Україні представлена як великими фабриками з потужною автоматизацією, так і малими чи середніми підприємствами, які діють локально. Усі ці підприємства мають справу з однаковим набором базових управлінських завдань: облік сировини, облік готової продукції, оформлення замовлень, контроль запасів, взаємодія з постачальниками та клієнтами. Проте рівень автоматизації процесів істотно різниться.

Малі кондитерські підприємства часто функціонують у ручному або напівручному режимі, в основному використовуючи звичайні Excel-таблиці, паперові документи або застарілі облікові програми без можливості мережевого доступу. При цьому вони стикаються з низкою викликів, серед яких:

- Відсутність централізованого обліку. Інформація про сировину, продажі, клієнтів зберігається в різних файлах або фізичних журналах, що ускладнює отримання актуальних зведень.
- Помилки в обліку залишків. Наприклад, при виробництві 100 штук тортів «Наполеон» із однієї партії сировини працівник не списав інгредієнти, і в системі залишається недостовірна кількість залишків. У результаті наступне замовлення може зірватись через брак борошна, про який не знали.
- Повільна обробка замовлень. Якщо інформація про клієнта та його замовлення передається зателефонувавши або через месенджер, а потім вручну переписується в зошит, це спричиняє помилки та втрату часу.
- Неможливість аналізу динаміки продажів або споживання сировини. Без цифрових звітів підприємець не може прогнозувати попит чи робити стратегічні закупки.

Для прикладу можна розглянути типову ситуацію, де міні-кондитерська обслуговує щодня до 30 клієнтів. При цьому використовується Google Sheets для запису замовлень, окремо ведеться зошит із закупівлями, а про залишки на складі власник дізнається від працівників вручну. Така схема працює, доки не виникає

перевантаження, особливо коли обсяг замовлень росте, помилки накопичуються, а працівники починають плутаються в рецептурах.

Ще одним поширеним прикладом є незручність у підрахунку вартості виробу на основі витрат сировини. Наприклад, для виготовлення партії тістечок потрібні інгредієнти в різних одиницях (кг, мл, шт.), але калькуляція робиться вручну, і часто не враховується вартість частково використаних компонентів (наприклад, 1/3 банки згущеного молока).

Також актуальною є проблема контролю залишків: якщо мінімальні запаси не контролюються, підприємство може виявити брак критичних інгредієнтів уже в момент початку роботи, наприклад, відсутність желатину або ванілі для святкового замовлення. Це може призвести до суттєвих репутаційних втрат.

Усе це свідчить про потребу в автоматизованій, адаптованій під специфіку малого бізнесу інформаційній системі, яка повинна забезпечити:

- облік сировини та продукції з прив’язкою до одиниць виміру;
- автоматичне списання інгредієнтів під час виготовлення партії;
- журнал замовлень із повним життєвим циклом, від прийняття до виконання;
- довідники клієнтів і постачальників;
- контроль критичних залишків на складі з попередженням;
- звітність за обсягами виробництва, витратами, продажами.

Звичайно, що розроблення такої система це не новина і на ринку вже давно існують такі системи, як BAS ERP, Poster POS та інші, однак вони часто вимагають щомісячної ліцензії або підтримки від сторонніх компаній. Окрім цього, вони часто мають складний інтерфейс і надмірну функціональність, що не потрібна малим підприємствам, оскільки потребує додаткового часу та ресурсів для навчання персоналу.

У відповідь на це виникає ідея створення власної клієнт-серверної системи, розробленої з урахуванням реальних потреб типового кондитерського підприємства середнього масштабу. Така система повинна бути простою, стабільною, працювати у локальній мережі, забезпечувати захист даних, а також дозволяти розширення, зокрема щодо додавання підтримки онлайн-замовлень у майбутньому.

1.2. Аналіз існуючих рішень

Для автоматизації облікових процесів у торгівлі та виробництві на ринку представлено чимало програмних продуктів різного рівня складності та вартості. Деякі з них орієнтовані на великі підприємства, інші на малий бізнес. У цьому розділі буде проаналізовано найбільш поширені рішення, які теоретично можуть бути використані в діяльності кондитерського підприємства.

BAS Малий бізнес / BAS Бухгалтерія. BAS (Business Automation Software) – це лінійка програм, що належать компанії «NetHelp» (Польща). BAS Бухгалтерія (рис.1.1) дозволяє вести фінансовий та податковий облік відповідно до законодавства України, тоді як BAS Малий бізнес включає елементи складського обліку, управління замовленнями, взаємодії з клієнтами.

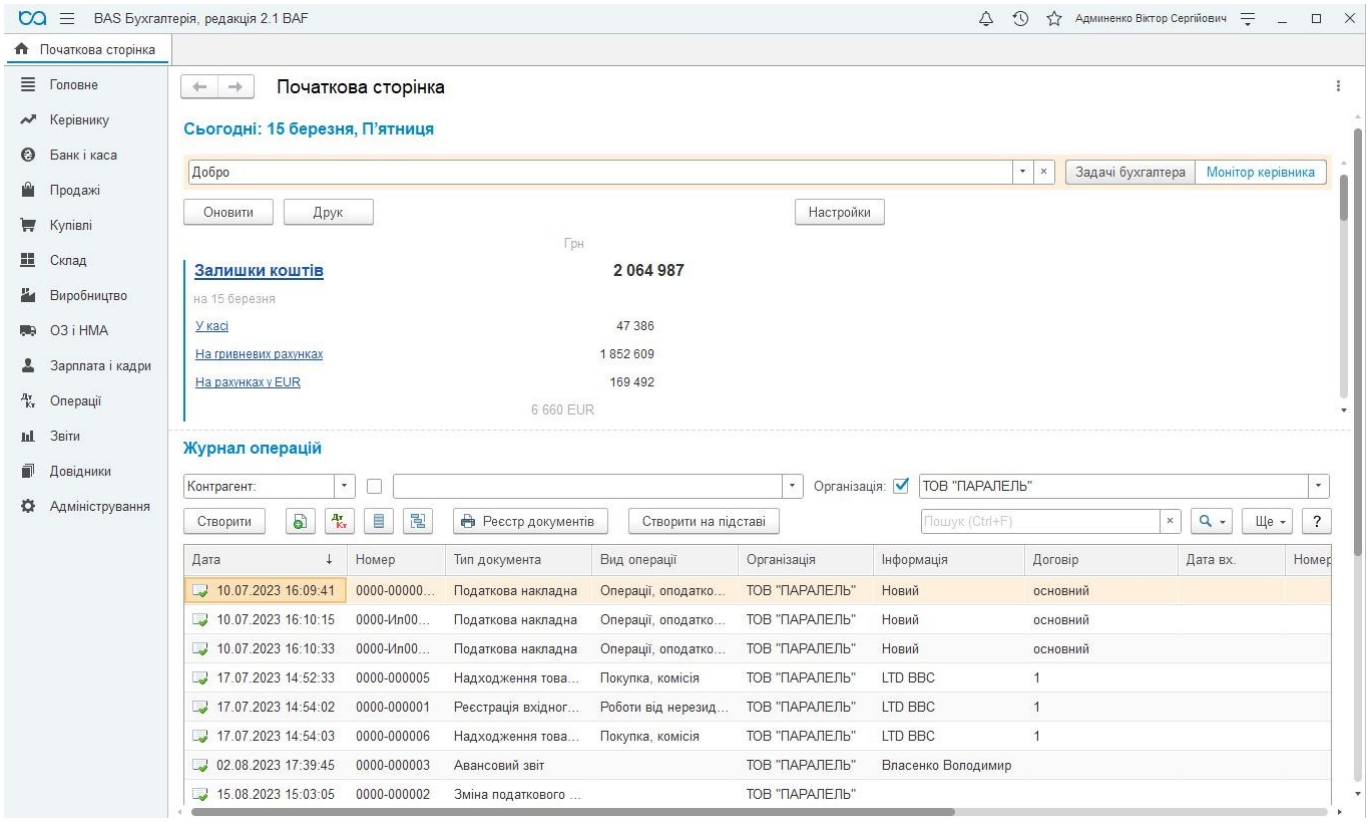

Рис.1.1 Вигляд графічного інтерфейсу програми BAS Бухгалтерія

До основних переваг цієї програми можна віднести повну відповідність всім вимогам бухгалтерії та податкового обліку, а також наявність модулів для обліку товарів, складу, взаєморозрахунків та підтримка звітності.

До недоліків можна сміливо віднести складний графічний інтерфейс, що вимагає навчання. Окрім цього, ця програма не призначена спеціально для

кондитерського виробництва, зокрема у ній відсутня деталізація рецептур. Ще одним суттєвим недоліком можна вважати наявну ліцензійну модель, яка передбачає доволі високу вартість обслуговування та оновлень. Загалом ця програма орієнтована більше на бухгалтера, ніж на виробника чи власника виробництва.

Poster POS. Poster це типова хмарна POS-система, популярна серед кафе, ресторанів, пекарень. Вона дозволяє вести облік продажів, залишків, створювати меню, обслуговувати замовлення через планшет (рис.1.2).

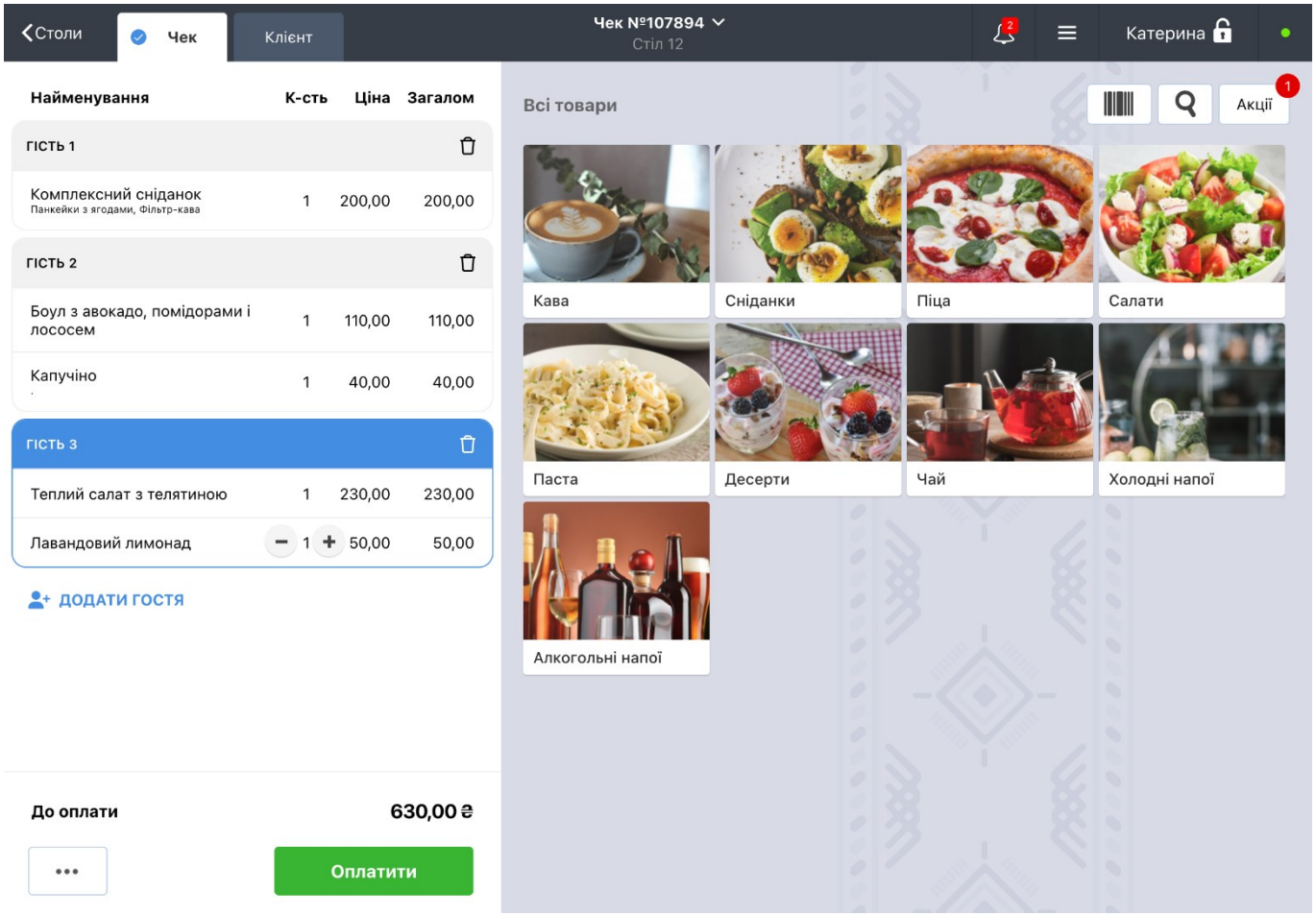

Рис.1.2 Вигляд графічного інтерфейсу програми Poster POS

До основних переваг цієї прогами можна віднести простоту та швидкість впровадження, зручний інтерфейс для касира а також наявність функцій обліку інгредієнтів, калькуляції та контролю залишків.

До недоліків можна віднести орієнтованість на сферу обслуговування, а не виробництво, залежність від стабільного інтернет-з'єднання (SaaS), щомісячна абонплата (від 300 грн на місяць), а також неможливість повної локалізації та зміни внутрішньої логіки.

Excel / Google Sheets. Досі одним із найпоширеніших способів ведення обліку серед невеликих виробників все ще залишаються електронні таблиці. До їх очевидних переваг можна віднести безкоштовність або низьку вартість, а також відносну простоту налаштування. Попри це, такий спосіб має ряд суттєвих недоліків до яких відноситься високий ризик людських помилок, відсутність автоматизації обліку замовлень, залишків. Окрім цього, такий підхід не підтримує авторизацію, обмеження доступу, журналювання дій а також не підходить для роботи в команді в офлайн-режимі.

Самописні локальні системи. Деякі підприємства замовляють власні програми у фріланс-розробників або дрібних студій. У цьому випадку до переваг можна віднести можливість індивідуального налаштування та адаптацію під конкретний бізнес-процес. Однак цей підхід має ряд суттєвих недоліків, серед яких в основному відсутність підтримки після завершення проєкту, складність масштабування, а також часто низький рівень документування та безпеки.

Таким чином, жодне з розглянутих рішень повністю не покриває потреби малого або середнього кондитерського підприємства, яке має унікальні особливості:

- виготовлення продукції за складними рецептурами;
- паралельна робота з сировиною та готовими виробами;
- необхідність ведення замовлень з контролем життєвого циклу;
- простий, зрозумілий інтерфейс для користувачів без технічної підготовки;
- мінімальні витрати на впровадження та підтримку.

У зв’язку з цим, виникає потреба у розробці індивідуального програмного забезпечення, яке базується на сучасних підходах, забезпечує автоматизацію ключових процесів, дозволяє локальну роботу без щомісячної абонплати та адаптоване під реалії вітчизняного малого виробництва.

1.3. Визначення функціональних і нефункціональних вимог до системи

На основі проведеного аналізу предметної області, а також вивчення існуючих аналогів, було сформульовано вимоги до інформаційної системи управління замовленнями та складським обліком для кондитерського підприємства. Ці вимоги

поділяються на функціональні, що визначають основні можливості системи, та нефункціональні, які описують характеристики якості програмного продукту.

Отже до функціональних вимог можна віднести:

Облік користувачів і ролей. Система повинна забезпечувати можливість створення облікових записів користувачів із різними рівнями доступу. Наприклад, адміністратор матиме повний контроль над системою, включно з налаштуваннями та управлінням користувачами, менеджер відповідатиме за оформлення та супровід замовлень, а складський працівник - за рух сировини. Це забезпечує контрольоване середовище використання системи та безпеку критичних даних. Також має бути реалізовано механізм ведення журналу подій - збереження історії дій користувачів.

Управління замовленнями. Однією з ключових функцій також має бути створення, редагування, перегляд та видалення замовлень. Кожне замовлення повинно містити дані про клієнта, список продукцій, дату оформлення, терміни виготовлення та доставки. Система має дозволяти змінювати статус замовлення (наприклад, «нове», «в роботі», «готово», «доставлено»), що дає змогу відслідковувати його поточний стан. Окрім цього, має бути можливість формувати супровідні документи, такі як накладна або рахунок.

Облік клієнтів. Система повинна зберігати інформацію про клієнтів, зокрема ім’я, контактні дані (телефон, email), історію замовлень. Це дозволяє налагодити повторний контакт із постійними покупцями, формувати персоналізовані пропозиції або скидки, а також ефективно вирішувати спірні питання, що виникають після продажу.

Складський облік. Необхідно реалізувати модуль обліку надходження сировини (борошно, цукор, яйця тощо) та її списання при виготовленні продукції згідно з рецептами. Система повинна вести точний облік залишків на складі, своєчасно повідомляючи відповідальних осіб про нестачу або перевищення критичних запасів. Це дозволить оптимізувати закупівлі та уникати перебоїв у виробництві.

Управління номенклатурою продукції. Програма повинна містити каталог продукції з характеристиками: назва виробу, категорія, ціна, склад, рецептура (список інгредієнтів та їх кількість на одиницю продукції). Під час обробки замовлення

система повинна автоматично розраховувати необхідну кількість сировини та списувати її зі складу. Це мінімізує помилки та підвищує точність виробництва.

Формування звітності. Для аналітики та прийняття управлінських рішень необхідна генерація наступних звітів:

- Звіт про кількість замовлень за період дозволить відслідковувати динаміку продажів.
- Звіт про використання сировини для оптимізації закупівлі.
- Звіт про прибутковість продукції та аналіз фінансових результатів.

Усі ці звіти мають виводитися на екран і бути доступними для експорту у форматі CSV та Excel.

У свою чергу, до нефункціональних вимог можна віднести:

Зручність використання (Usability). Інтерфейс системи повинен бути інтуїтивно зрозумілим, орієнтованим на користувачів без спеціальної підготовки. Всі основні дії мають бути доступними через кнопки, меню та діалогові вікна. Усі підписи, повідомлення й інструкції повинні бути виключно українською мовою, що підвищує доступність для працівників підприємства.

Надійність і безпека. Система повинна зберігати дані в захищеному вигляді. Паролі користувачів мають зберігатися з використанням хешування. Передбачено функцію резервного копіювання бази даних, щоб запобігти втраті інформації. Всі критичні дії (видалення, редагування) повинні бути обмежені для певних ролей і зафіксовані у журналі подій.

Продуктивність. Система повинна ефективно працювати з великими обсягами даних, зокрема з базою даних, що містить тисячі записів. Час відгуку інтерфейсу має бути мінімальним навіть при навантаженні, що забезпечить комфортну роботу користувачів у реальному часі.

Масштабованість. Система повинна бути спроєктована з урахуванням можливості подальшого розширення: додавання нових модулів (наприклад, онлайн-замовлення), інтеграції з зовнішніми сервісами (наприклад, Google Calendar, Telegram), або перенесення бази даних на сервер. Архітектура має підтримувати багатокористувацький режим у локальній мережі підприємства.

Платформенні обмеження. На першому етапі реалізації передбачається розроблення десктопної версії під операційну систему Windows. Система повинна працювати на звичайних офісних комп'ютерах без необхідності встановлення складного додаткового програмного забезпечення. Зберігання даних реалізується за допомогою локального серверу бази даних MySQL або SQLite.

Передбачається, що користувачі мають базову комп'ютерну грамотність. На першому етапі система не буде мати онлайн-доступу, а працюватиме лише у межах локальної мережі. Рецептури виробів вважаються постійними, тобто не змінюються під час експлуатації системи. Взаємодія з податковими або бухгалтерськими системами не передбачається. Усі документи, що формуються системою, мають рекомендаційний характер і не є юридично зобов'язуючими.

1.4. Обгрунтування вибору архітектури та засобів реалізації

При розробленні інформаційної системи управління замовленнями та складським обліком для кондитерського підприємства ключовим завданням стало обрання архітектурного підходу та відповідних інструментів реалізації, що забезпечать ефективність, зручність, масштабованість і надійність розроблюваного програмного забезпечення.

З огляду на вимоги до багатокористувацького доступу, централізованого зберігання даних, можливості мережевої взаємодії та подальшої підтримки, було прийнято рішення про використання клієнт-серверної архітектури. Такий підхід дозволяє розділити функціональність системи на два логічних рівні: клієнтський та серверний. Сервер відповідає за зберігання, обробку та цілісність даних, тоді як клієнтська частина надає зручний інтерфейс користувачу для взаємодії з системою. Такий розподіл забезпечує централізований контроль за даними, покращує безпеку доступу та реалізацію розмежування прав користувачів за ролями.

Як мову програмування для створення клієнтської частини було обрано C#, що входить до складу платформи .NET від Microsoft. Цей вибір обґрунтований кількома факторами: по-перше, C# забезпечує високу продуктивність і є одним з найбільш зручних та потужних засобів для розробки настільних застосунків. По-друге, він

підтримує об’єктно-орієнтований підхід, що дозволяє створювати гнучку та модульну архітектуру програми. По-третє, C# має тісну інтеграцію з численними бібліотеками та фреймворками, зокрема для роботи з базами даних та інтерфейсом користувача.

Графічний інтерфейс користувача реалізовуватиметься за допомогою бібліотеки Windows Forms, яка є стабільним і перевіреним інструментом для побудови віконних застосунків у середовищі Windows. Ця бібліотека дозволяє швидко створювати багатовіконні інтерфейси з кнопками, таблицями, формами вводу, меню, повідомленнями та іншими елементами керування. У порівнянні з більш сучасною технологією WPF, WinForms має меншу складність у реалізації та достатній функціонал для середньої складності бізнес-додатків.

Для зберігання даних обрано реляційну СКБД MySQL. Вона являє собою найпопулярнішу і найстабільнішу СКБД з відкритим вихідним кодом, що широко застосовується у прикладних системах малого та середнього бізнесу. MySQL підтримує створення складних зв’язків між таблицями, цілісність даних, транзакції, запити з агрегацією та обробкою великих обсягів інформації. Окрім того, вона дозволяє організувати мережевий доступ до серверу бази даних, що критично важливо для клієнт-серверної архітектури. Зв’язок клієнтської частини з MySQL буде реалізований через офіційну бібліотеку MySql.Data, яка дозволяє виконувати SQL-запити та обробляти результати у форматі, зручному для подальшої роботи у C#.

Розроблення програмного забезпечення буде здійснюватись у середовищі Microsoft Visual Studio, яке є найбільш функціональним та зручним інструментом для створення застосунків на C#. Воно підтримує інтелектуальне автозаповнення коду, налагодження, дизайн UI у візуальному редакторі та має гнучку інтеграцію з системами контролю версій. Для створення та адміністрування структури бази даних буде використано MySQL Workbench, що дозволяє створювати ER-діаграми, писати та виконувати SQL-скрипти, а також здійснювати резервне копіювання бази.

Обрані технології у сукупності забезпечують швидку розробку, високу підтримуваність, зручність для користувачів та ефективність функціонування всієї системи. Крім того, така конфігурація дозволяє надалі масштабувати програму зокрема шляхом перенесення системи на віддалений хмарний сервер.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Інформаційна модель системи

Інформаційна модель системи визначає структуру та взаємозв'язки об'єктів предметної області, які обробляються програмним забезпеченням. Вона є основою для проєктування бази даних і програмної логіки взаємодії користувача з системою. Для управління замовленнями та складським обліком на кондитерському підприємстві були виділені наступні ключові сутності:

Користувач (User), який представляє співробітника підприємства, який має обліковий запис у системі. Загалом ця сутність складається з наступних атрибутів:

- *UserID (PK)*: унікальний ідентифікатор користувача (цілочисельне автоінкрементне поле).
- *Login:* Унікальний логін для входу до системи.
- *PasswordHash:* Хеш пароля користувача (не зберігається у відкритому вигляді).
- *Role:* Роль користувача (адміністратор, менеджер, комірник), визначає рівень доступу.
- *FullName:* Повне ім'я користувача для ідентифікації в інтерфейсі.
- *Email:* Адреса електронної пошти (для зв'язку, повідомлень).
- *Phone:* Номер телефону користувача.
- *IsActive:* Ознака активності облікового запису (*true* - активний, *false* - це заблокований).

Клієнт (Client). Суб'єкт, що замовляє продукцію підприємства (фізична або юридична особа). Ця сутність складається з:

- *ClientID (PK)*: Унікальний ідентифікатор клієнта.
- *Name:* Назва або ПІБ клієнта.
- *Type:* Тип клієнта (фізична або юридична особа).
- *Phone:* Контактний номер телефону.
- *Email:* Адреса електронної пошти.
- *Address:* Повна поштова адреса клієнта.

Постачальник (Supplier). Організація або особа, яка постачає сировину або матеріали для виробництва. Зокрема тут можна виділити:

- *SupplierID (PK)*: Унікальний ідентифікатор постачальника.
- *Name:* Назва постачальника.
- *ContactPerson:* ПІБ контактної особи.
- *Phone:* Телефон для зв'язку.
- *Email:* Електронна адреса постачальника.
- *Address:* Адреса місцезнаходження.

Складський товар (StockItem). Описує одиницю обліку на складі, наприклад сировину або готову продукцію. Тут варто відзначити:

- *ItemID (PK):* Унікальний ідентифікатор товару.
- *Name:* Назва товару (наприклад, «Борошно пшеничне», «Торт "Наполеон"»).
- *Type:* Тип (сировина або продукція).
- *Unit:* Одиниця виміру (кг, л, шт. тощо).
- *Quantity:* Поточна кількість на складі.
- *MinThreshold:* Мінімальний рівень залишку, при якому система формує сповіщення.

Рецептура (Recipe). Описує склад продукту з прив'язкою до інгредієнтів. До цієї сутності можна віднести наступні атрибути:

- *RecipeID (PK):* Унікальний ідентифікатор запису рецептури.
- *ProductID (FK → StockItem):* Ідентифікатор готового продукту.
- *IngredientID (FK → StockItem):* Ідентифікатор інгредієнту.
- *QuantityRequired:* Кількість інгредієнту, потрібна для виготовлення однієї одиниці продукції.

Замовлення (Order). Це одна з найважливіших сутностей, яка містить загальну інформацію про окреме замовлення клієнта, зокрема:

- *OrderID (PK):* Унікальний ідентифікатор замовлення.
- *ClientID (FK):* Замовник, пов'язаний із сутністю Client.
- *UserID (FK):* Працівник, який створив або обробляє замовлення.
- *OrderDate:* Дата створення замовлення.

- *DueDate:* Запланована дата виконання.
- *Status:* Статус замовлення (нове, виконано, скасовано).
- *TotalAmount:* Загальна вартість замовлення.

Позиція замовлення (OrderItem). Кожен рядок замовлення, що відповідає конкретному продукту, зокрема:

- *OrderItemID (PK):* Унікальний ідентифікатор позиції.
- *OrderID (FK):* Посилання на замовлення.
- *ProductID (FK):* Ідентифікатор продукції (готовий виріб).
- *Quantity:* Кількість одиниць.
- *Price:* Ціна за одиницю.
- *Subtotal:* Підсумкова вартість (Quantity × Price).

Операція складу (StockOperation). Запис про зміну кількості товарів на складі. Ця сутність містить наступні атрибути:

- *OperationID (PK):* Унікальний ідентифікатор операції.
- *ItemID (FK):* Товар, який було змінено.
- *OperationType:* Тип операції (Надходження, Списання, Витрата).
- *Quantity:* Кількість, на яку змінено запас.
- *Date:* Дата виконання операції.
- *RelatedOrderID (FK, NULL):* Пов’язане замовлення (опційно).
- *UserID (FK):* Користувач, який виконав дію.

Журнал дій (AuditLog). Фіксує всі важливі дії користувачів у системі. По суті ця сутність являє собою запис логів і буде містити такі атрибути:

- *LogID (PK):* Унікальний ідентифікатор запису.
- *UserID (FK):* Користувач, який виконав дію.
- *Timestamp:* Дата та час дії.
- *Action:* Тип дії (вхід, редагування, видалення, створення).
- *Details:* Текстовий опис або JSON-дані із деталями.

Сповіщення (Notification). Повідомлення користувачам про події системи. Тут можна віднест наступні атрибути:

- *NotificationID (PK):* Унікальний ідентифікатор повідомлення.

- *UserID (FK):* Отримувач повідомлення.
- *Message:* Текст повідомлення.
- *DateCreated:* Дата створення повідомлення.
- *IsRead:* Ознака, чи було прочитано повідомлення.

Ця інформаційна модель дозволяє реалізувати чітке логічне представлення всіх об'єктів і процесів підприємства. На її основі можливо ефективно керувати обігом продукції, взаємодією з клієнтами, закупівлями, контролем залишків та звітністю, що є критично важливим для стабільної роботи кондитерського виробництва.

2.2. Математична модель процесів обліку замовлень

Математична модель системи описує основні обчислення, логіку перетворення даних і алгоритми, які забезпечують роботу бізнес-процесів підприємства. Вона є основою для автоматизованого управління запасами, виконанням замовлень, обчислення витрат, контролю залишків і формування звітів.

Загалом цю модель можна розділити на кілька складових, серед яких:

Модель управління замовленнями. Давайте приймемо, що загальна вартість замовлення може бути розрахована таким чином:

$$S(z_i) = \sum_{j=1}^{m} q_{ij} \cdot c_{ij} \quad (2.1)$$

Де:

$Z = \{z_1, z_2, \ldots, z_n\}$ – множина замовлень;

$z_i = (k_i, P_i, q_i, c_i, d_i)$;

k_i – клієнт, який здійснив замовлення;

$P_i = \{p_{i1}, p_{i2}, ..., p_{im}\}$ – множина продуктів у замовленні;

q_{ij} – кількість одиниць продукту p_{ij};

c_{ij} – ціна одиниці продукту p_{ij};

d_i – дата замовлення.

Цей розрахунок використовується при створенні, редагуванні та фіналізації замовлень, а також при генерації фінансової звітності.

Модель обліку складу. Якщо уявити, що $T = \{t_1, t_2, ..., t_k\}$ – множина складських товарів, тоді $t_j = (n_j, u_j, q_j, q_{min,j})$, де:

n_j – назва товару;

u_j – одиниця виміру (шт., кг, л);

q_j – поточна кількість на складі;

$q_{min,j}$ – мінімально допустимий рівень запасів.

Водночас, математична умова для формування попередження може бути записана наступним чином:

$$q_j < q_{min,j} \Rightarrow \text{сформувати сповіщення про критичний запас} \quad (2.2)$$

Це правило реалізується як тригер при оновленні складу або періодичному скануванні залишків.

Модель рецептури (виробництва продукції). Виробництво кожної одиниці продукції потребує певної кількості сировини. Нехай:

$$R = \{(p_k, s_1, a_1), ..., (p_k, s_m, a_m)\} \quad (2.3)$$

де:

p_k – продукт;

s_i – інгредієнт (товар типу «сировина»);

a_i – кількість інгредієнта si, що потрібна для виробництва однієї одиниці продукту p_k.

При виконанні замовлення z_i із q одиниць продукту p_k, система розраховує необхідну кількість кожного інгредієнта:

$$Q_{si} = ai \cdot q \quad (2.4)$$

Це значення списується зі складу автоматично в момент виконання замовлення.

Операції руху товарів. Система підтримує облік усіх складських операцій, зокрема двох основних:

Надходження:

$$q_j := q_j + \Delta \quad (2.5)$$

Витрата (списання):

$$qj := qj - \Delta \quad (2.6)$$

Варто також зазначити, що всі ці операції можна класифікувати за типом, зокрема це може бути постачання, внутрішнє використання, виробництво, повернення, знищення тощо.

Звітність та аналітика. Формуються агреговані значення для обчислення:

- Обсягу продажів за період:

$$S_{total}(T) = \sum_{zi \epsilon T} S(z_i) \quad (2.7)$$

- Витрат матеріалів за період:

$$V_{sj}(T) = \sum_{\text{усіх замовлень}} a_j \cdot q_{pk} \quad (2.8)$$

- Рівень залишків:

$$Z_s = q_j \quad (2.9)$$

Таким чином, математична модель забезпечує точність обліку, обробку замовлень, контроль складських запасів, а також автоматизує типові аналітичні процеси. Вона є критичною для впровадження надійної інформаційної системи управління підприємством.

2.3. Проєктування структури бази даних

Для реалізації інформаційної системи управління замовленнями та складським обліком було спроєктовано реляційну базу даних, яка відповідає інформаційній моделі, описаній у попередньому підрозділі. Основна мета проєктування бази даних полягала у забезпеченні надійного зберігання, цілісності та швидкого доступу до інформації, яка використовується під час функціонування клієнт-серверної системи.

Загалом, база даних будується відповідно до нормалізованої моделі, де всі сутності мають чіткі первинні ключі (PK) і зв’язки між собою реалізовані через зовнішні ключі (FK), зокрема було використано типові зв’язки один до багатьох (1:N) для таких пар, як:

Client → Order (клієнт має багато замовлень);

Order → OrderItem (одне замовлення містить кілька позицій);

User → Order (один користувач створює багато замовлень);

StockItem → Recipe (один продукт може мати кілька інгредієнтів);

User → AuditLog, StockOperation, Notification.

У випадку таблиці *Recipe* фактично реалізується зв’язок багато до багатьох (M:N) між готовою продукцією та інгредієнтами, який нормалізовано в окрему таблицю з додатковим атрибутом, а саме кількістю інгредієнту.

Нижче наведено опис структури бази даних, вказано основні типи полів для реалізації в СУБД MySQL. Зокрема на початку необхідно створити саму базу даних. Для цього використовується запит:

```
CREATE DATABASE OrderStockDB CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE OrderStockDB;
```

Цей запит створює базу даних із підтримкою повноцінного Юнікоду (utf8mb4), що забезпечує коректну роботу з українськими символами.

Для зберігання облікових записів користувачів системи, таких як адміністратори, менеджери та комірники, створюється таблиця *Users*:

```
CREATE TABLE Users (
    UserID INT AUTO_INCREMENT PRIMARY KEY,
    Login VARCHAR(50) UNIQUE NOT NULL,
    PasswordHash VARCHAR(255) NOT NULL,
    Role ENUM('Адміністратор', 'Менеджер', 'Комірник') NOT NULL,
    FullName VARCHAR(100) NOT NULL,
    Email VARCHAR(100),
    Phone VARCHAR(20),
    IsActive BOOLEAN DEFAULT TRUE
);
```

Для зберігання даних про клієнтів: як фізичних, так і юридичних осіб, необхідно створити таблицю *Clients:*

```
CREATE TABLE Clients (
    ClientID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Type ENUM('Фізична особа', 'Юридична особа') NOT NULL,
    Phone VARCHAR(20),
    Email VARCHAR(100),
    Address TEXT
);
```

Наступна таблиця *Suppliers* зберігає інформацію про постачальників сировини або продукції:

```
CREATE TABLE Suppliers (
    SupplierID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    ContactPerson VARCHAR(100),
    Phone VARCHAR(20),
    Email VARCHAR(100),
    Address TEXT
);
```

У свою чергу, таблиця *StockItems* містить перелік усіх товарних позицій: сировини та готової продукції.

```
CREATE TABLE StockItems (
    ItemID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Type ENUM('Сировина', 'Продукція') NOT NULL,
    Unit VARCHAR(20) NOT NULL,
    Quantity DECIMAL(10,2) NOT NULL DEFAULT 0,
    MinThreshold DECIMAL(10,2) DEFAULT 0
);
```

Водночас для визначення рецептури виготовлення продукції, зокрема які інгредієнти і в якій кількості потрібні, створено таблицю *Recipes*:

```
CREATE TABLE Recipes (
    RecipeID INT AUTO_INCREMENT PRIMARY KEY,
    ProductID INT NOT NULL,
    IngredientID INT NOT NULL,
    QuantityRequired DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (ProductID) REFERENCES StockItems(ItemID) ON DELETE CASCADE,
    FOREIGN KEY (IngredientID) REFERENCES StockItems(ItemID) ON DELETE CASCADE
);
```

Таблиця *Orders* містить замовлення клієнтів, зокрема інформацію про те хто, коли і що замовив.

```
CREATE TABLE Orders (
    OrderID INT AUTO_INCREMENT PRIMARY KEY,
    ClientID INT NOT NULL,
    UserID INT NOT NULL,
    OrderDate DATE NOT NULL,
    DueDate DATE,
    Status ENUM('Нове', 'Виконано', 'Скасовано') DEFAULT 'Нове',
    TotalAmount DECIMAL(10,2),
    FOREIGN KEY (ClientID) REFERENCES Clients(ClientID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Ще однією важливою таблицею є *OrderItems*, яка містить деталі до замовлень, зокрема конкретні позиції, кількість і ціна.

```
CREATE TABLE OrderItems (
    OrderItemID INT AUTO_INCREMENT PRIMARY KEY,
    OrderID INT NOT NULL,
    ProductID INT NOT NULL,
    Quantity DECIMAL(10,2) NOT NULL,
```

```
    Price DECIMAL(10,2) NOT NULL,
    Subtotal DECIMAL(10,2) GENERATED ALWAYS AS (Quantity * Price) STORED,
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID) ON DELETE CASCADE,
    FOREIGN KEY (ProductID) REFERENCES StockItems(ItemID)
);
```

Наступна таблиця *StockOperations* фіксує всі зміни кількості товарів на складі: надходження, списання, витрати.

```
CREATE TABLE StockOperations (
    OperationID INT AUTO_INCREMENT PRIMARY KEY,
    ItemID INT NOT NULL,
    OperationType ENUM('Надходження', 'Списання', 'Витрата') NOT NULL,
    Quantity DECIMAL(10,2) NOT NULL,
    Date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    RelatedOrderID INT,
    UserID INT,
    FOREIGN KEY (ItemID) REFERENCES StockItems(ItemID),
    FOREIGN KEY (RelatedOrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Водночас таблиця *AuditLog* необхідна для логування дій користувачів у системі для контролю змін та безпеки.

```
CREATE TABLE AuditLog (
    LogID INT AUTO_INCREMENT PRIMARY KEY,
    UserID INT NOT NULL,
    Timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Action VARCHAR(50) NOT NULL,
    Details TEXT,
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Остання таблиця *Notifications* містить повідомлення користувачам про важливі події або зміни.

```
CREATE TABLE Notifications (
    NotificationID INT AUTO_INCREMENT PRIMARY KEY,
    UserID INT NOT NULL,
    Message TEXT NOT NULL,
    DateCreated DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    IsRead BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

На рисунку 2.1 представлено логічну модель даних у вигляді схеми «множина–сутність–зв’язок» (ER-моделі). Для її побудови було адаптовано структуру бази даних у форматі DBML, що є сумісним із платформою dbdiagram.io, яка й використовувалась для візуалізації моделі.

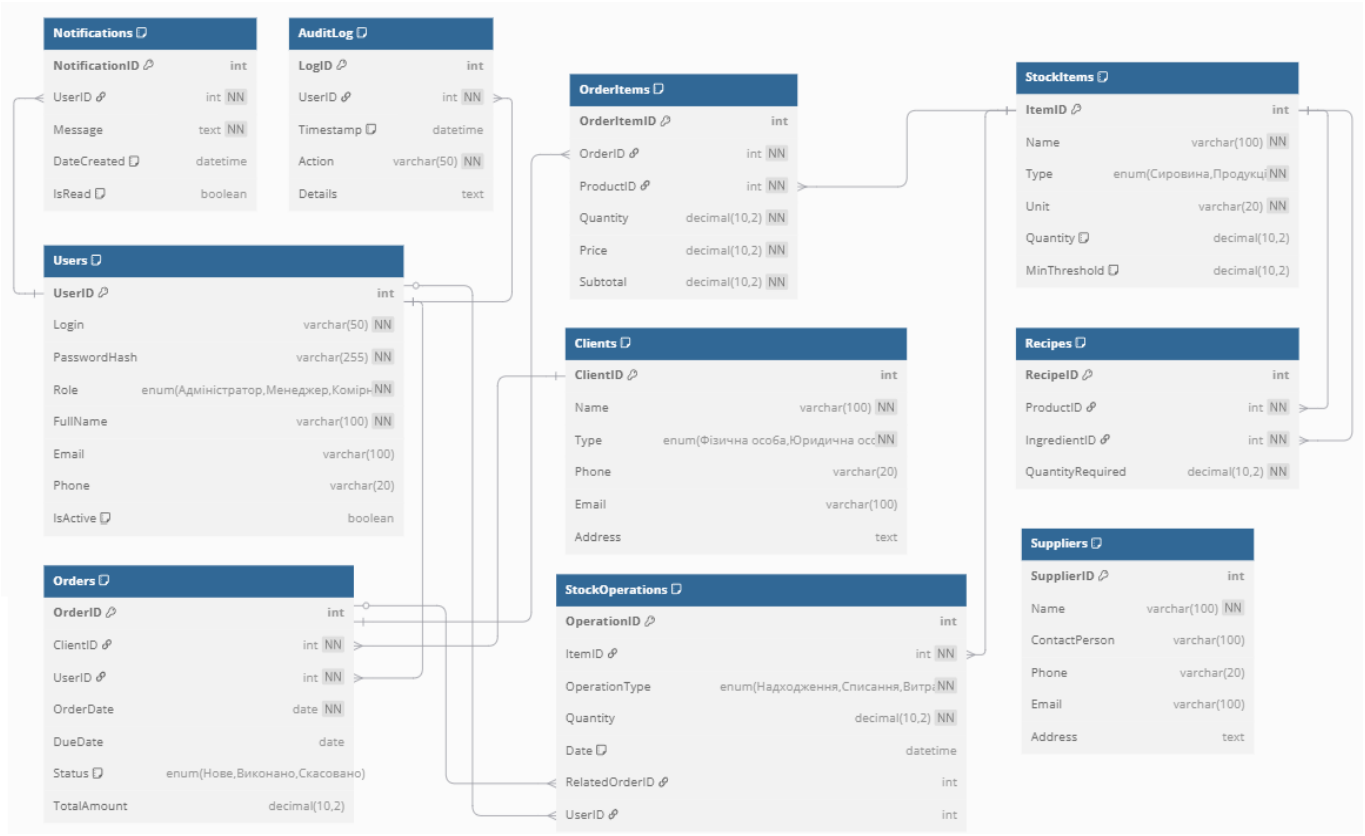

Рис.2.1 Вигляд розробленої ER-діаграми створеної БД

2.4. Розмежування доступу та логіка ролей користувачів

У клієнт-серверній інформаційній системі, що автоматизує управління замовленнями та складським обліком на кондитерському підприємстві, надзвичайно важливо забезпечити належний рівень безпеки та контроль дій користувачів. Для цього реалізовано механізм розмежування доступу на основі ролей (Role-Based Access Control, RBAC), що дозволяє обмежити або надати доступ до функціональних можливостей системи відповідно до посадових обов’язків користувачів.

Загалом передбачено три ролі користувачів, зокрема:

- *Адміністратор* має повний доступ до всієї функціональності системи. Може створювати, редагувати та видаляти користувачів, переглядати журнали дій, а також конфігурувати системні параметри.

- *Менеджер* відповідає за створення та обробку замовлень, перегляд інформації про клієнтів і продукцію, генерацію звітів, але не має доступу до змін у базовій конфігурації системи чи користувачах.
- *Комірник* здійснює контроль залишків на складі, оперує з операціями надходження/списання товарів, оновлює фактичні запаси, але не має доступу до розділів замовлень та клієнтів.

Після проходження процедури аутентифікації (входу до системи з використанням унікального логіна та хешованого пароля) кожному користувачеві призначається роль відповідно до значення атрибуту *Role* у таблиці *Users*. На основі цього визначається рівень доступу до окремих модулів і функцій клієнтського застосунку.

У таблиці 2.1 наведено реалізацію контролю доступу на рівні основних операцій системи.

Табл.2.1

Контроль доступу на рівні основних операцій системи

Функціональність	Адміністратор	Менеджер	Комірник
Створення / редагування користувачів	ТАК	НІ	НІ
Робота із замовленнями	ТАК	ТАК	НІ
Операції над складом	ТАК	НІ	ТАК
Генерація звітів	ТАК	ТАК	НІ
Перегляд журналу дій	ТАК	НІ	НІ

Контроль доступу реалізовано на рівні клієнтського застосунку засобами мови C# у поєднанні з умовною логікою та механізмом блокування елементів інтерфейсу для неавторизованих ролей. У разі спроби доступу до забороненого функціоналу система виводить відповідне повідомлення, а подія фіксується у журналі аудиту (*AuditLog*). Окрім цього, у структурі бази даних збережено атрибут *IsActive*, який дозволяє тимчасово блокувати користувача без видалення облікового запису.

Отже при вході в систему користувач вводить логін і пароль. Пароль не зберігається у відкритому вигляді, його хеш завжди обчислюється алгоритмом SHA-256, і лише цей хеш порівнюється з тим, що зберігається в базі даних. Зокрема цей підхід вдалося реалізувати наступним чином:

```
public class AuthService
{
    public User Authenticate(string login, string password)
    {
        using (var db = new AppDbContext())
        {
            string hash = HashPassword(password);
            return db.Users
                     .FirstOrDefault(u => u.Login == login
                                     && u.PasswordHash == hash
                                     && u.IsActive);
        }
    }

    private string HashPassword(string password)
    {
        using (var sha = SHA256.Create())
        {
            var bytes = sha.ComputeHash(Encoding.UTF8.GetBytes(password));
            return Convert.ToBase64String(bytes);
        }
    }
}
```

У свою чергу, у таблиці *Users* роль кожного користувача визначається атрибутом *Role*. Ця інформація зчитується при вході й використовується для визначення рівня доступу, зокрема використано наступний SQL запит:

```
SELECT Role
FROM Users
WHERE Login = 'ivanov' AND IsActive = TRUE;
```

Водночас після авторизації, доступ користувача обмежується відповідно до його ролі. Це досягається шляхом блокування або приховування частин інтерфейсу:

```
public void ApplyRolePermissions(User user)
{
    switch (user.Role)
    {
        case "Адміністратор":
            // Повний доступ
            break;

        case "Менеджер":
            adminTab.Enabled = false;
            stockTab.Enabled = false;
            break;

        case "Комірник":
            ordersTab.Enabled = false;
            clientsTab.Enabled = false;
            adminTab.Enabled = false;
            break;
    }
}
```

Цей підхід забезпечує зручність використання системи та мінімізує ризик помилок, оскільки користувач бачить лише ті функції, які йому дозволено виконувати.

Варто також зазначити, що для забезпечення безпеки системи та можливості подальшого аналізу передбачено логування всіх важливих дій користувачів у таблиці AuditLog. Запис включає користувача, тип дії, дату й опис події, зокрема:

```
public void LogUserAction(int userId, string action, string details)
{
    using (var db = new AppDbContext())
    {
        var log = new AuditLog
        {
            UserID = userId,
            Timestamp = DateTime.Now,
            Action = action,
            Details = details
        };

        db.AuditLogs.Add(log);
        db.SaveChanges();
    }
}
```

Ці дані можуть бути використані як для виявлення несанкціонованих дій, так і для внутрішнього контролю виконання операцій працівниками підприємства.

Таким чином, розмежування доступу за ролями забезпечує як безпеку, так і зручність у роботі з системою. Усі основні дії користувачів фіксуються в журналі, а рівень доступу визначається при авторизації та обмежується на рівні як бази даних, так і інтерфейсу.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Середовище розробки та мова програмування

Для реалізації клієнт-серверного програмного забезпечення було обрано сучасний стек технологій, що дозволяє забезпечити надійність, масштабованість та простоту супроводу програмного коду. Основним середовищем розробки стала Microsoft Visual Studio 2022, яка надає потужний набір інструментів для створення, налагодження та тестування програм з графічним інтерфейсом, роботою з базами даних і використанням технологій клієнт-серверної архітектури.

В якості основної мови програмування використано C#, яка є сучасною об'єктно-орієнтованою мовою з широкими можливостями для побудови десктопних застосунків, взаємодії з базами даних, реалізації REST API, а також реалізації систем з високим рівнем безпеки та продуктивності. Платформа .NET 6 була обрана як основа для серверної логіки та API, оскільки вона є кросплатформенною, активно підтримується Microsoft та має високі показники продуктивності.

Клієнтська частина програми реалізована з використанням Windows Forms, тобто перевіреної технології для створення графічних інтерфейсів на базі .NET. Вона дозволяє швидко реалізувати зручний та функціональний інтерфейс з доступом до бази даних і зовнішніх API.

Серверна частина базується на шаблоні Web API, реалізованому за допомогою ASP.NET Core. Цей підхід дозволяє розділити логіку обробки даних і клієнтську взаємодію, створити зручний API для масштабування системи, а також забезпечити можливість інтеграції з іншими сервісами в майбутньому.

Для роботи з базою даних застосовано ORM-бібліотеку Entity Framework Core, яка спрощує доступ до даних та дозволяє працювати з базою через об’єктно-орієнтовані моделі. Усі запити до бази формуються у вигляді LINQ-виразів, що покращує читабельність коду і зменшує ризик SQL-ін’єкцій.

Таким чином, обрані інструменти та мова програмування забезпечують ефективну реалізацію вимог до програмного забезпечення з урахуванням продуктивності, безпеки, зручності розробки та можливості масштабування в майбутньому.

3.2. Реалізація клієнтської частини застосунку

Одним із важливих етапів проєктування клієнтської частини застосунку є побудова діаграми класів, яка відображає логічну структуру взаємозв'язків між компонентами системи та їх основні характеристики. Ця діаграма дозволяє візуалізувати архітектуру клієнтської частини з погляду об'єктно-орієнтованого підходу: класи, методи, атрибути, а також їхні залежності та асоціації.

Діаграма класів створена відповідно до загальної логіки поведінки клієнтського застосунку, яка реалізована за допомогою мови програмування C# з використанням технології Windows Forms. Основними функціональними модулями клієнтської частини є:

- модулі авторизації та аутентифікації користувачів,
- модулі взаємодії з серверними API, які забезпечують доступ до бази даних,
- форми інтерфейсу користувача для роботи з замовленнями, користувачами, складом та звітністю.

Зокрема у діаграмі представлено ключові класи:

User, який описує модель користувача із полями облікового запису та роллю;

Order, як модель для замовлень клієнтів;

ApiService, що реалізує обробку запитів до серверної частини (створення, отримання, оновлення та видалення об'єктів);

AuthService, що відповідає за авторизацію користувачів;

LoginForm, MainForm, OrdersForm, UsersForm – графічні форми Windows Forms, через які персонал взаємодіє із системою.

Завдяки поділу відповідальностей між класами досягається високий рівень модульності, що спрощує подальше тестування, супровід і розширення функціоналу. Наприклад, логіка перевірки прав доступу реалізується окремо в сервісах, що дозволяє повторно використовувати їх у різних формах.

Таким чином, на рисунку 3.1 наведено діаграму класів клієнтської частини застосунку, яка охоплює моделі, сервіси та інтерфейсні компоненти.

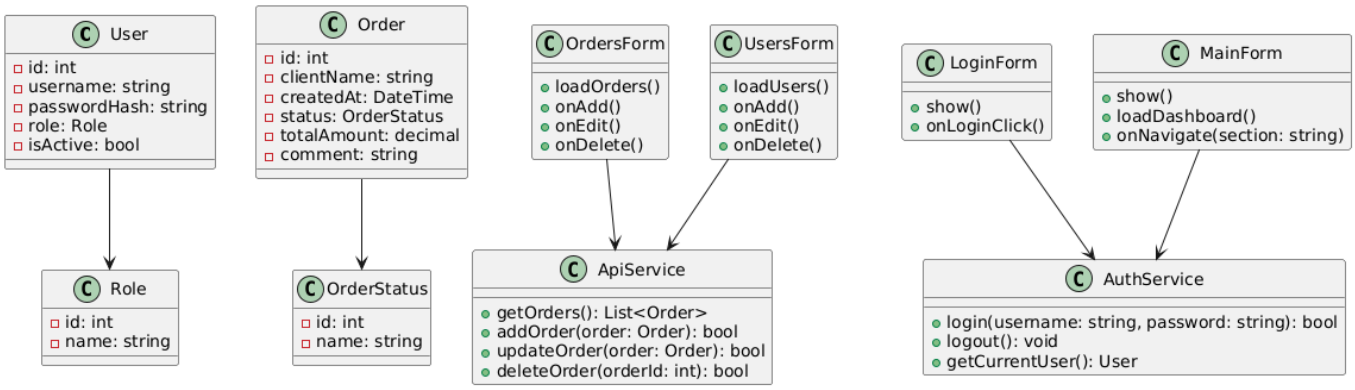

Рис.3.1 Вигляд розробленої діаграми класів UML

Клієнтська частина програмного забезпечення реалізована у вигляді десктопного застосунку з використанням технології Windows Forms. Цей підхід забезпечує швидку розробку інтерфейсу, зручність використання для користувача та просту інтеграцію з серверною частиною через HTTP-запити.

Інтерфейс програми складається з окремих форм, що відповідають різним функціональним модулям: авторизації, обліку замовлень, управління користувачами, перегляду звітів тощо. Між формами здійснюється навігація через обробку подій кнопок. Таким чином можна виділити наступні розроблені форми:

Форма авторизації (рис. 3.2) є першою точкою взаємодії користувача із системою. Вона містить два текстові поля для введення логіна та пароля, а також кнопку «Увійти». Після введення даних, вони передаються на сервер. Якщо автентифікація успішна, користувач переходить до головного вікна. У разі помилки виводиться повідомлення з поясненням: «Невірний логін або пароль».

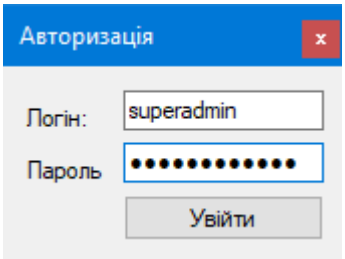

Рис.3.2 Вікно авторизації користувача

Після входу в систему користувач потрапляє до головного вікна (рис.3.3), яке містить панель навігації та меню з доступними функціональними модулями. В залежності від ролі користувача, деякі кнопки (наприклад, «Користувачі», «Звіти») можуть бути неактивними або прихованими.

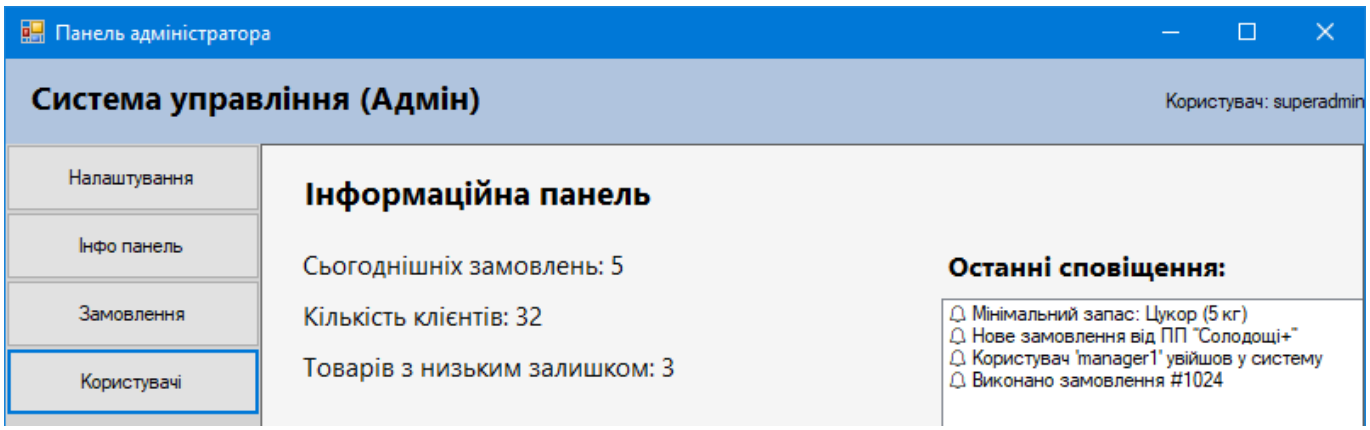

Рис.3.3.1 Головне вікно клієнтського застосунку (адміністратор)

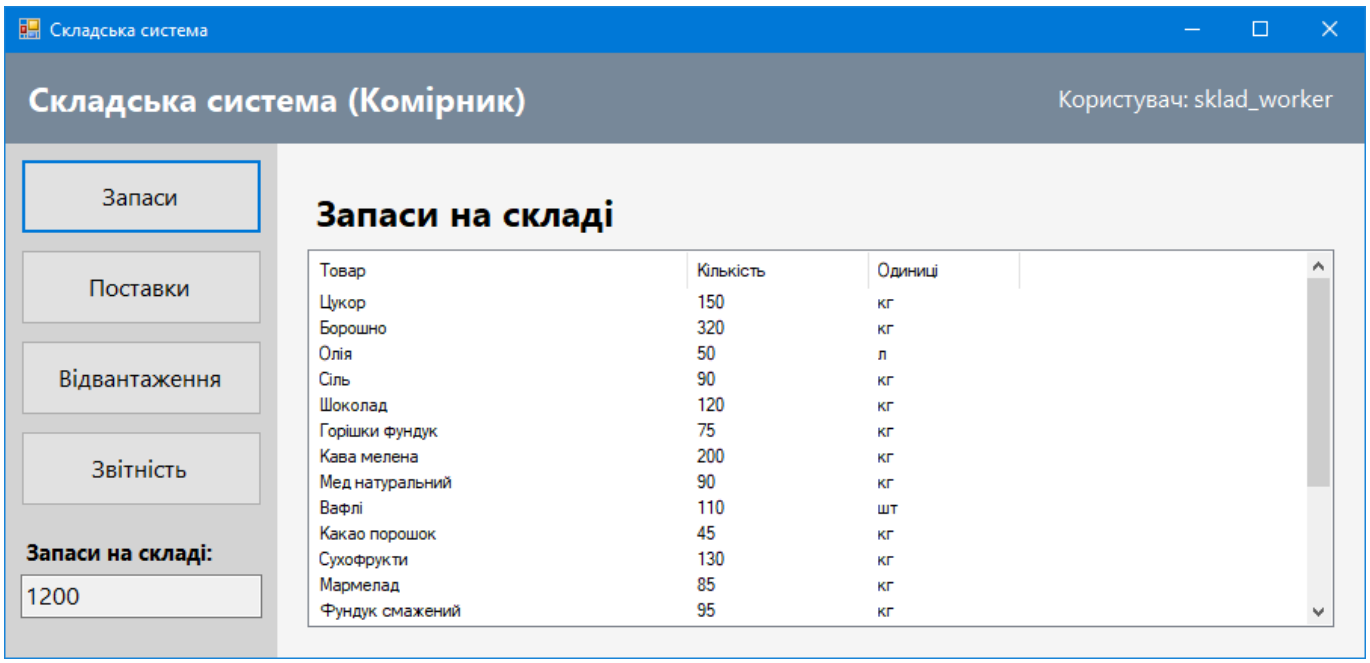

Рис.3.3.2 Головне вікно клієнтського застосунку (комірник)

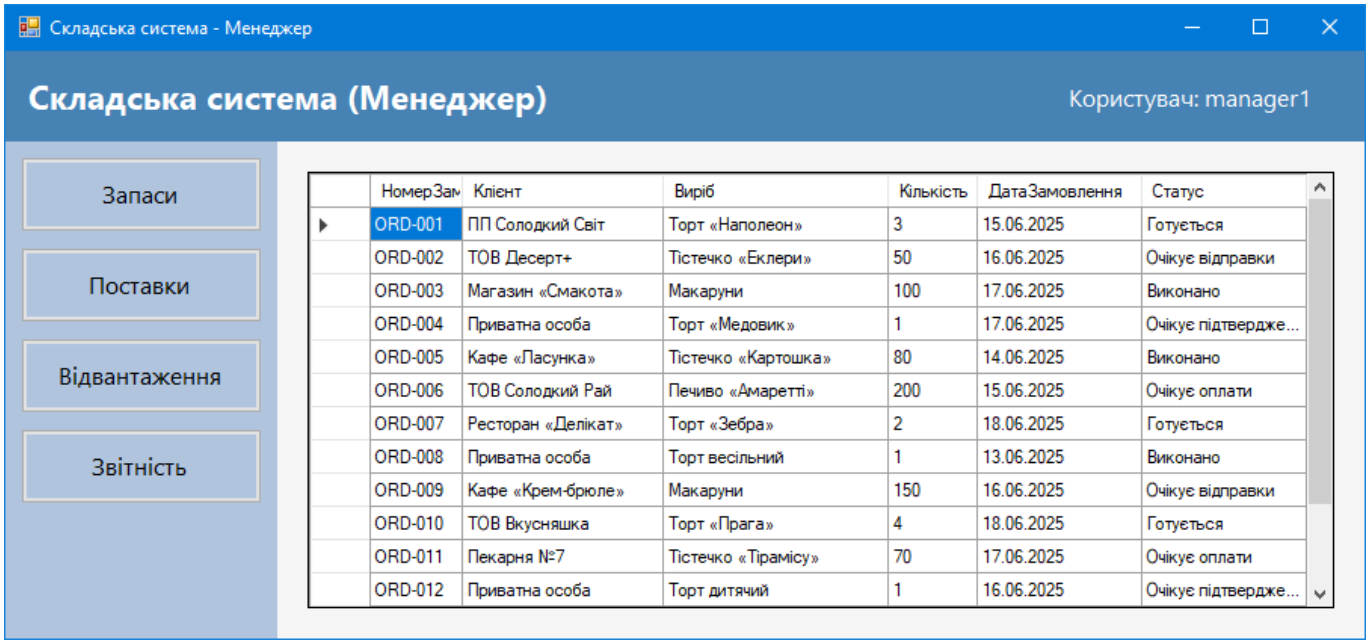

Рис.3.3.3 Головне вікно клієнтського застосунку (менеджер)

До особливостей реалізації можна віднести динамічне формування інтерфейсу відповідно до ролі користувача; кнопки для переходу до підрозділів: «Замовлення», «Користувачі», «Вихід»; а також інформація про поточного користувача.

Ще однією з основних форм, яка дозволяє переглядати, додавати, редагувати та видаляти записи про замовлення є форма управління замовленнями (рис.3.4). Основним компонентом цієї форми є DataGridView, який динамічно заповнюється даними із сервера. Доступна також фільтрація за датами, статусами та пошук за клієнтом або номером замовлення.

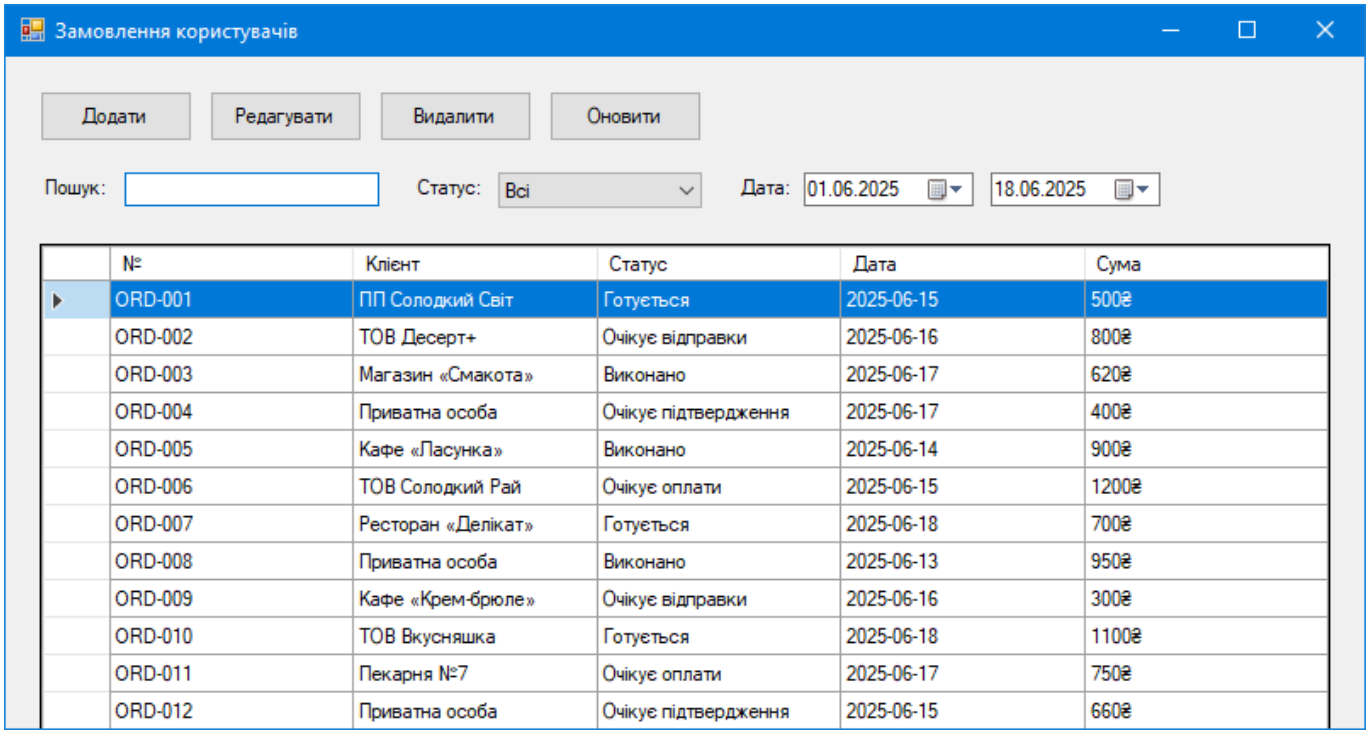

Рис.3.4 Перегляд та обробка замовлень

У даному випадку для особливостей реалізації форми можна віднести: асинхронне завантаження замовлень; модальні вікна для редагування і створення; підтвердження перед видаленням; а також індикація статусу замовлення, зокрема кольорове маркування.

Наступна форма управління користувачами (рис.3.5) доступна лише системним адміністраторам і призначена для створення, редагування та видалення облікових записів. У таблиці відображено список усіх зареєстрованих користувачів, їх логіни, ролі, статуси активності. Також реалізовано форму для створення нового користувача з валідацією введених даних.

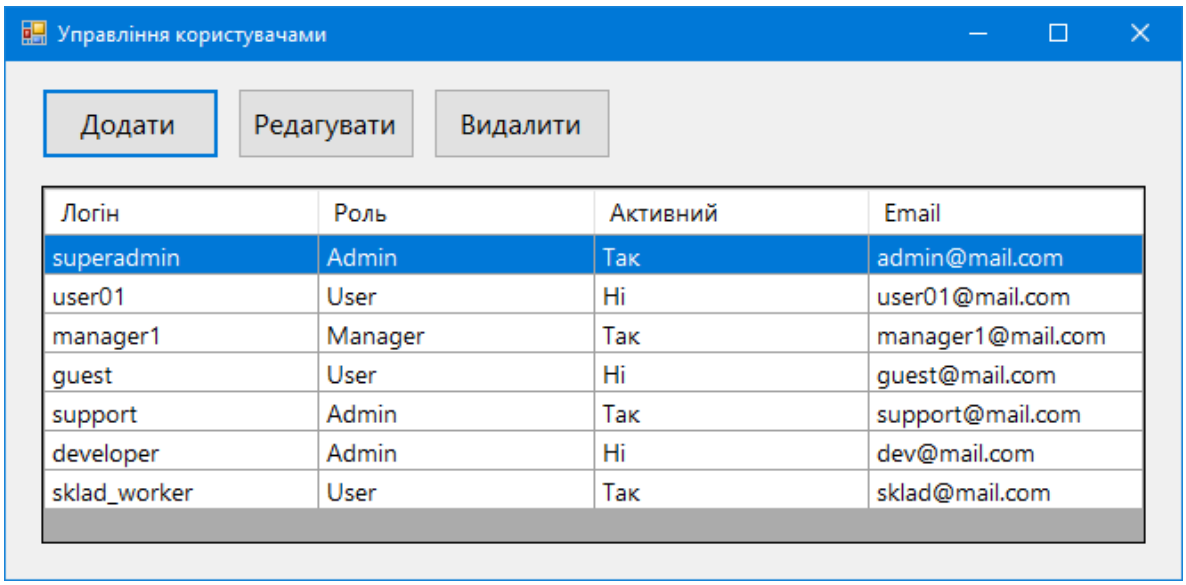

Рис.3.5 Інтерфейс керування користувачами

До основних особливостей реалізації цієї форми можна віднести: встановлення та зміна ролі (admin/user/worker); активація/деактивація облікового запису; перевірка унікальності логіна; обмеження доступу до цієї форми.

У свою чергу, при натисканні кнопки «Додати» або «Редагувати» у таблицях замовлень чи користувачів відкриваються окремі модальні форми з формами введення. Після заповнення даних і натискання кнопки «Зберегти» відбувається відправка SQL-запиту на сервер. Таким чином, форма редагування запису (рис.3.6) має ряд особливостей, зокрема перевірка обов'язкових полів; перевірка форматів (наприклад, дати або електронної пошти); повідомлення про успішне збереження.

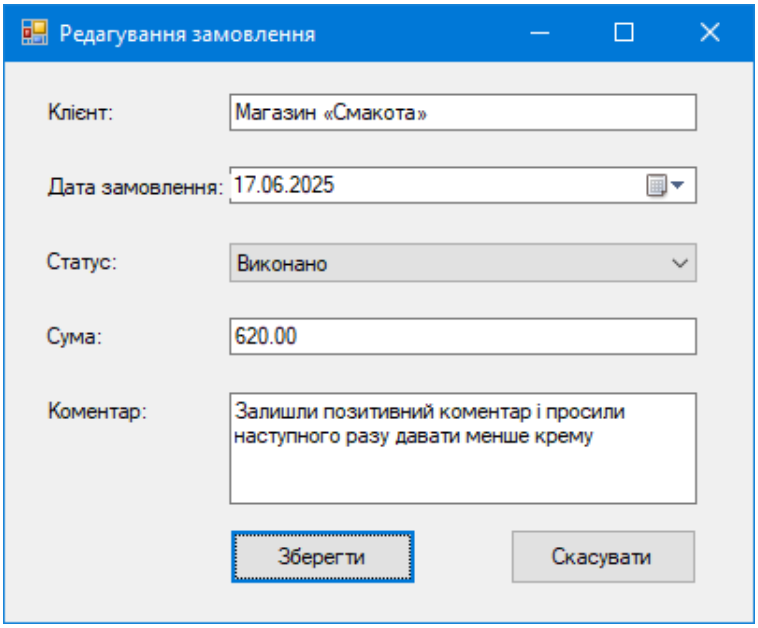

Рис.3.6 Форма редагування запису

Варто зазначити, що весь застосунок реалізує валідацію введення на стороні клієнта до відправки на сервер. У разі виявлення помилок виводяться відповідні підказки: зокрема, червона рамка навколо поля або повідомлення в MessageBox.

Водночас, для спрощення роботи з API реалізовано окремий сервіс *ApiService.cs*, який містить методи *GetAsync, PostAsync, PutAsync, DeleteAsync*. Це дозволяє централізовано обробляти всі запити, повідомлення про помилки та логіку повторних спроб.

3.3. Організація серверної взаємодії та доступу до бази даних

Серверна взаємодія в контексті настільного застосунку реалізована без використання HTTP-запитів або REST API. Уся логіка зосереджена всередині C# застосунку, який безпосередньо звертається до бази даних, розгорнутої на сервері MySQL через MySQL Workbench. З'єднання з базою даних відбувається за допомогою бібліотеки MySql.Data.MySqlClient.

Щоб підключитися до бази даних, застосунок використовує рядок підключення наступного вигляду:

```
string connectionString =
"Server=localhost;Database=mydatabase;Uid=myuser;Pwd=mypassword;";
```

У свою чергу для створення з’єднання використано запит:

```
using MySqlConnection connection = new MySqlConnection(connectionString);
connection.Open();
```

Під час авторизації, користувач вводить логін і пароль у відповідну форму. Ці дані перевіряються за допомогою SQL-запиту до таблиці *users*:

```
string query = "SELECT role FROM users WHERE username = @username AND password =
SHA2(@password, 256)";
using MySqlCommand command = new MySqlCommand(query, connection);
command.Parameters.AddWithValue("@username", inputUsername);
command.Parameters.AddWithValue("@password", inputPassword);

object role = command.ExecuteScalar();
if (role != null)
{
    // Успішний вхід, збереження ролі користувача
    currentUserRole = role.ToString();
}
else
{
    MessageBox.Show("Невірний логін або пароль");
}
```

Варто зазначити, що паролі у базі зберігаються лише у хешованому вигляді за допомогою алгоритму SHA2, що забезпечує базовий рівень безпеки.

Водночас, при створенні нового запису, наприклад, додавання товару в базу, формується наступний запит:

```
string insertQuery = "INSERT INTO products (name, quantity, price) VALUES (@name,
@quantity, @price)";
using MySqlCommand cmd = new MySqlCommand(insertQuery, connection);
cmd.Parameters.AddWithValue("@name", nameTextBox.Text);
cmd.Parameters.AddWithValue("@quantity", int.Parse(quantityTextBox.Text));
cmd.Parameters.AddWithValue("@price", double.Parse(priceTextBox.Text));
cmd.ExecuteNonQuery();
```

У свою чергу, отримання даних для DataGridView, зокрема при завантаженні таблиці замовлень, використовується такий запит:

```
string query = "SELECT id, client_name, order_date FROM orders";
using MySqlDataAdapter adapter = new MySqlDataAdapter(query, connection);
DataTable dt = new DataTable();
adapter.Fill(dt);
ordersGridView.DataSource = dt;
```

У залежності від ролі, яку повернула система після входу, застосунок динамічно вмикає або вимикає доступ до певних частин інтерфейсу:

```
if (currentUserRole == "Admin")
{
    adminPanel.Visible = true;
}
else
{
    adminPanel.Visible = false;
}
```

Таке розмежування доступу є важливим засобом безпеки й логічного контролю. Наприклад, адміністратор має доступ до редагування таблиць, створення звітів, а користувач лише до їх перегляду.

Варто також зазначити, що кожна дія з БД супроводжується обробкою винятків для уникнення аварійного завершення програми:

```
try
{
    connection.Open();
    // виконати запит
}
catch (MySqlException ex)
{
    MessageBox.Show("Помилка бази даних: " + ex.Message);
}
finally
{
    connection.Close();
}
```

Таким чином, у даній реалізації серверна логіка і доступ до бази даних реалізовані без проміжного веб-сервера. Клієнтський застосунок безпосередньо звертається до БД через SQL-запити, які формуються динамічно відповідно до дій користувача. Такий підхід є класичним варіантом побудови клієнта з прямим підключенням до бази даних.

3.4. Тестування функціоналу системи

Тестування є невід'ємною частиною життєвого циклу ПЗ. У межах реалізованого застосунку було проведено модульне тестування окремих методів та функцій, а також інтеграційне тестування взаємодії компонентів.

Зокрема модульне тестування включало:

- Перевірку обробки порожніх або некоректних полів при авторизації.
- Валідацію введених даних (числові значення для кількості або цін).
- Перевірку результату виконання SQL-запитів та їх впливу на базу даних.

Наприклад модульний тест для перевірки авторизації наступний:

```
public void TestLoginWithWrongCredentials()
{
    var result = AuthService.TryLogin("wronguser", "wrongpass");
    Assert.IsFalse(result);
}
```

Водночас, інтеграційне тестування перевіряло:

- Зв'язок між формами.
- Чи відображаються зміни в базі даних у відповідних елементах інтерфейсу.
- Коректність сценаріїв використання – створення, редагування, видалення записів.

Щодо валідації введення, то усі поля, пов'язані з введенням тексту, чисел або дат, проходять перевірку на допустимість значень. Це дозволяє уникнути введення SQL-коду або некоректних даних, що можуть викликати помилки на рівні БД.

```
if (string.IsNullOrWhiteSpace(nameTextBox.Text))
{
    MessageBox.Show("Ім'я не може бути порожнім");
    return;
}
```

У свою чергу для забезпечення безпеки при взаємодії з MySQL застосовано параметризовані запити, які виключають можливість SQL-ін'єкцій:

```
string query = "SELECT * FROM users WHERE username = @username AND password =
SHA2(@password, 256)";
command.Parameters.AddWithValue("@username", login);
command.Parameters.AddWithValue("@password", password);
```

Це гарантує, що введені користувачем значення не можуть змінити логіку запиту. При цьому, що стосується захисту паролів, то вони зберігаються виключно у хешованому вигляді за допомогою алгоритму SHA-256. Це забезпечує, що навіть у разі витоку бази даних паролі не зберігаються у відкритому вигляді:

```
UPDATE users SET password = SHA2('newpass', 256) WHERE id = 1;
```

Окремо варто відзначити реалізацію обмеження прав доступу. Як зазначалося раніше, у системі реалізовано ролеву модель доступу:

- Адміністратор: має повний доступ до всіх функцій системи.
- Користувач: доступ лише до перегляду й базових операцій.
- Комірник: мінімальний рівень – лише робота із складською інформацією.

Усі обмеження реалізовано на рівні логіки програми, тобто відповідні функції або кнопки стають недоступними або невидимими в залежності від ролі.

```
if (currentUserRole != "Admin")
{
    btnDelete.Visible = false;
    btnEdit.Enabled = false;
}
```

Також варто зазначити про захист даних та резервне копіювання. Зокрема дані в базі зберігаються у таблицях з унікальними ключами, що виключає дублювання. Додатково реалізовано механізм експорту даних до CSV, що дозволяє створювати резервні копії. Приклад такого SQL-експорту:

```
SELECT * FROM orders
INTO OUTFILE '/tmp/orders_backup.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Окрім цього, передбачено захист на рівні підключення до БД, зокрема у застосунку не зберігаються облікові дані до бази у відкритому вигляді. Рядок підключення може зберігатися у зашифрованому конфігураційному файлі або в окремому шифрованому сховищі. Також здійснюється обмеження доступу до БД лише з локального хоста або з IP-адреси сервера програми.

ВИСНОВКИ

У процесі виконання дипломної роботи було розроблено клієнт-серверну інформаційну систему, призначену для автоматизації процесів управління замовленнями та обліку товарів на складі кондитерського підприємства. Розробка охоплювала усі основні етапи життєвого циклу ПЗ від постановки задачі та аналізу предметної області до реалізації, тестування та захисту даних.

Система базується на архітектурі клієнт-сервер із централізованою базою даних, що дозволяє забезпечити одночасний доступ декількох користувачів до актуальної інформації у межах єдиного інформаційного простору. Клієнтська частина реалізована у вигляді десктопного застосунку мовою C# із використанням Windows Forms, що забезпечує зручну багатовіконну взаємодію з користувачем. Серверна частина побудована на основі СУБД MySQL, яка забезпечує надійне зберігання та обробку інформації.

Функціональні можливості створеної системи охоплюють облік готової продукції, визначення її складу з інгредієнтів, ведення довідників постачальників, клієнтів, сировини, а також реєстрацію та обробку замовлень із подальшим автоматичним списанням продукції та матеріалів зі складу. Крім цього, реалізовано оновлення залишків, контроль критичних запасів, а також формування звітності за замовленнями, витратами, обсягами продажів та залишками на складі за визначені періоди.

Особливу увагу приділено питанням безпеки та контролю доступу. Система підтримує механізм авторизації користувачів з розмежуванням прав доступу згідно з визначеними ролями (адміністратор, менеджер, комірник). Для збереження конфіденційності реалізовано шифрування паролів, а також ведення журналу дій користувачів з метою фіксації критичних операцій у системі.

Таким чином, поставлені у технічному завданні цілі дипломної роботи було досягнуто повною мірою. Система продемонструвала свою працездатність під час тестування, відповідає функціональним вимогам, є зручною у використанні та придатною до подальшого впровадження на реальному виробництві.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Albahari, J., & Albahari, B. (2021). C# 10 in a Nutshell: The Definitive Reference. O'Reilly Media.

2. Esposito, D. (2014). Programming Microsoft ASP.NET MVC. Microsoft Press.

3. Freeman, A., & Sanderson, P. (2020). Pro ASP.NET Core MVC 2. Apress.

4. Liberman, Y. (2016). Pro MySQL. Apress.

5. Anderson, J. (2013). SQL for MySQL Developers: A Comprehensive Tutorial and Reference. Addison-Wesley.

6. Price, J. (2018). Microsoft Visual C# Step by Step (9th ed.). Microsoft Press.

7. Sadalage, P. J., & Fowler, M. (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley.

8. Microsoft Docs. (n.d.). Role-based security in .NET. Retrieved June 16, 2025, from https://learn.microsoft.com/

9. Dev.to. (2023). Implementing role-based access control in C# applications. Retrieved from https://dev.to/steven/implementing-role-based-access-control-in-csharp-2023

10. MySQL Documentation. (n.d.). MySQL 8.0 Reference Manual - User Account Management. Retrieved from https://dev.mysql.com/doc/

11. Medium. (2022). Building a Role-Based Access Control System using MySQL and .NET. Retrieved from https://medium.com/@rbac/mysql-csharp-rbac

12. GitHub. (2024). Inventory Management System using C# and MySQL. Retrieved from https://github.com/example/inventory-management-csharp

13. DB-Engines. (2023). MySQL - Popularity and features. Retrieved from https://db-engines.com/en/system/MySQL

14. Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. IEEE Computer, 29(2), 38–47. https://doi.org/10.1109/2.485845

15. He, Q., Daudjee, K., & Salem, K. (2019). Fine-grained access control for relational databases. ACM Transactions on Database Systems, 44(3), 1–44. https://doi.org/10.1145/3331480

Додаток А - Вигляд системних логів

LogID	UserID	Timestamp	Action	Details
1	2	2025-06-14 08:01:02	login	Користувач увійшов у систему
2	2	2025-06-14 08:03:50	create_client	{ "ClientID": 201, "Name": "ПП 'Медовик'", "Type": "фізична особа" }
3	2	2025-06-14 08:04:20	create_client	{ "ClientID": 202, "Name": "ТОВ 'Солодко і Крапка'", "Type": "юридична особа" }
4	3	2025-06-14 09:10:43	create_order	{ "OrderID": 3001, "ClientID": 201, "TotalAmount": 1750.00 }
5	3	2025-06-14 09:11:12	add_order_item	{ "OrderID": 3001, "ProductID": 12, "Quantity": 5, "Price": 350.00 }
6	3	2025-06-14 09:11:55	add_order_item	{ "OrderID": 3001, "ProductID": 14, "Quantity": 2, "Price": 250.00 }
7	4	2025-06-14 10:00:21	stock_out	{ "ItemID": 7, "Quantity": 4.0, "OperationType": "Витрата", "RelatedOrderID": 3001 }
8	4	2025-06-14 10:10:32	stock_out	{ "ItemID": 8, "Quantity": 3.0, "OperationType": "Витрата", "RelatedOrderID": 3001 }
9	4	2025-06-14 11:00:00	stock_in	{ "ItemID": 3, "Quantity": 100.0, "OperationType": "Надходження", "SupplierID": 5 }
10	4	2025-06-14 11:45:03	stock_writeoff	{ "ItemID": 6, "Quantity": 1.5, "OperationType": "Списання", "Reason": "Псування" }
11	2	2025-06-14 12:00:00	update_recipe	{ "RecipeID": 9, "Updated": true, "Changes": { "IngredientID": 3, "QuantityRequired": 0.3 } }
12	2	2025-06-14 12:30:15	create_user	{ "UserID": 6, "Role": "менеджер", "FullName": "Ольга Демчук" }
13	1	2025-06-14 13:00:05	block_user	{ "UserID": 7, "Reason": "Три невдалі спроби входу" }
14	2	2025-06-14 13:30:55	update_user_profile	{ "UserID": 3, "FieldChanged": "Phone", "NewValue": "+380931112233" }
15	3	2025-06-14 14:20:10	change_order_status	{ "OrderID": 3001, "NewStatus": "виконано" }
16	2	2025-06-14 14:30:00	create_notification	{ "NotificationID": 17, "Message": "Замовлення №3001 виконано" }
17	3	2025-06-14 14:35:20	create_order	{ "OrderID": 3002, "ClientID": 202, "TotalAmount": 3200.00 }
18	3	2025-06-14 14:35:44	add_order_item	{ "OrderID": 3002, "ProductID": 11, "Quantity": 10, "Price": 320.00 }
19	3	2025-06-14 14:36:02	add_order_item	{ "OrderID": 3002, "ProductID": 9, "Quantity": 3, "Price": 250.00 }
20	4	2025-06-14 15:00:12	stock_out	{ "ItemID": 5, "Quantity": 15.0, "OperationType": "Витрата", "RelatedOrderID": 3002 }
21	4	2025-06-14 15:45:22	stock_in	{ "ItemID": 2, "Quantity": 25.0, "OperationType": "Надходження", "SupplierID": 2 }
22	4	2025-06-14 16:30:18	stock_out	{ "ItemID": 8, "Quantity": 6.0, "OperationType": "Витрата", "RelatedOrderID": 3002 }

LogID	UserID	Timestamp	Action	Details
23	3	2025-06-14 17:00:00	change_order_status	{ "OrderID": 3002, "NewStatus": "виконано" }
24	2	2025-06-14 17:10:30	create_supplier	{ "SupplierID": 12, "Name": "ТОВ 'БорошноПлюс'" }
25	2	2025-06-14 17:25:10	update_client	{ "ClientID": 202, "FieldChanged": "Address", "NewValue": "м. Львів, вул. Хлібна, 5" }
26	1	2025-06-14 17:30:55	delete_user	{ "UserID": 9, "Reason": "Звільнення" }
27	2	2025-06-14 18:00:00	update_password	{ "UserID": 3, "ChangedBy": "admin" }
28	4	2025-06-14 18:10:20	stock_writeoff	{ "ItemID": 4, "Quantity": 2.5, "Reason": "Закінчився термін придатності" }
29	3	2025-06-14 18:15:00	create_order	{ "OrderID": 3003, "ClientID": 201, "TotalAmount": 1100.00 }
30	3	2025-06-14 18:15:35	add_order_item	{ "OrderID": 3003, "ProductID": 16, "Quantity": 4, "Price": 275.00 }
31	4	2025-06-14 18:30:10	stock_out	{ "ItemID": 8, "Quantity": 5.0, "OperationType": "Витрата", "RelatedOrderID": 3003 }
32	2	2025-06-14 19:00:00	logout	Користувач вийшов із системи
33	2	2025-06-15 08:00:00	login	Користувач увійшов у систему
34	2	2025-06-15 08:05:15	create_user	{ "UserID": 10, "Role": "комірник", "FullName": "Іван Слюсар" }
35	2	2025-06-15 08:10:00	assign_role	{ "UserID": 10, "Role": "комірник" }
36	3	2025-06-15 08:40:00	update_order_item	{ "OrderID": 3003, "ProductID": 16, "Quantity": 5 }
37	4	2025-06-15 09:00:00	stock_in	{ "ItemID": 9, "Quantity": 60.0, "SupplierID": 6 }
38	1	2025-06-15 09:15:22	view_logs	{ "Filter": "UserID=3; Date=2025-06-14" }
39	2	2025-06-15 09:45:40	create_notification	{ "Message": "Рівень залишків по ItemID=4 нижчий за норму" }
40	2	2025-06-15 10:00:00	logout	Користувач вийшов із системи

Додаток Б - Фрагменти програмного коду

Реєстрація користувача

```
public bool RegisterUser(string login, string fullName, string email, string phone, string
password, string role)
{
    using (var connection = new MySqlConnection(connectionString))
    {
        string hashedPassword = BCrypt.Net.BCrypt.HashPassword(password);

        string query = @"
            INSERT INTO Users (Login, FullName, Email, Phone, PasswordHash, Role, IsActive)
            VALUES (@Login, @FullName, @Email, @Phone, @PasswordHash, @Role, TRUE)";

        using (var command = new MySqlCommand(query, connection))
        {
            command.Parameters.AddWithValue("@Login", login);
            command.Parameters.AddWithValue("@FullName", fullName);
            command.Parameters.AddWithValue("@Email", email);
            command.Parameters.AddWithValue("@Phone", phone);
            command.Parameters.AddWithValue("@PasswordHash", hashedPassword);
            command.Parameters.AddWithValue("@Role", role); // має бути: "Адміністратор",
"Менеджер" або "Комірник"

            connection.Open();
            int result = command.ExecuteNonQuery();
            return result > 0;
        }
    }
}
```

Вхід користувача

```
public bool LoginUser(string login, string password)
{
    using (var connection = new MySqlConnection(connectionString))
    {
        string query = "SELECT PasswordHash FROM Users WHERE Login = @Login AND IsActive = TRUE";
        using (var command = new MySqlCommand(query, connection))
        {
            command.Parameters.AddWithValue("@Login", login);
            connection.Open();
            var reader = command.ExecuteReader();

            if (reader.Read())
            {
                string hash = reader.GetString(0);
                return BCrypt.Net.BCrypt.Verify(password, hash);
            }
            return false;
        }
    }
}
```

Отримання інформації про користувача після входу

```
public class User
```

```
{
    public int UserID { get; set; }
    public string Login { get; set; }
    public string Email { get; set; }
    public string Role { get; set; }
    public string FullName { get; set; }
    public bool IsActive { get; set; }
}

public User GetUserData(string login)
{
    using (var connection = new MySqlConnection(connectionString))
    {
        string query = @"SELECT UserID, Login, Email, Role, FullName, IsActive
                         FROM Users
                         WHERE Login = @Login";

        using (var command = new MySqlCommand(query, connection))
        {
            command.Parameters.AddWithValue("@Login", login);
            connection.Open();

            using (var reader = command.ExecuteReader())
            {
                if (reader.Read())
                {
                    return new User
                    {
                        UserID = reader.GetInt32("UserID"),
                        Login = reader.GetString("Login"),
                        Email = reader.IsDBNull(reader.GetOrdinal("Email")) ? null :
reader.GetString("Email"),
                        Role = reader.GetString("Role"),
                        FullName = reader.GetString("FullName"),
                        IsActive = reader.GetBoolean("IsActive")
                    };
                }
                return null;
            }
        }
    }
}
```

Додавання нового замовлення

```
public int AddOrder(int clientId, int userId, DateTime orderDate, DateTime? dueDate, string
status, decimal totalAmount)
{
    using (var connection = new MySqlConnection(connectionString))
    {
        string query = @"
            INSERT INTO Orders (ClientID, UserID, OrderDate, DueDate, Status, TotalAmount)
            VALUES (@ClientID, @UserID, @OrderDate, @DueDate, @Status, @TotalAmount);
            SELECT LAST_INSERT_ID();";

        using (var command = new MySqlCommand(query, connection))
        {
            command.Parameters.AddWithValue("@ClientID", clientId);
```

```
            command.Parameters.AddWithValue("@UserID", userId);
            command.Parameters.AddWithValue("@OrderDate", orderDate);
            command.Parameters.AddWithValue("@DueDate", dueDate.HasValue ? dueDate.Value :
(object)DBNull.Value);
            command.Parameters.AddWithValue("@Status", status);
            command.Parameters.AddWithValue("@TotalAmount", totalAmount);

            connection.Open();
            return Convert.ToInt32(command.ExecuteScalar()); // Повертає ID нового замовлення
        }
    }
}
```

Редагування існуючого замовлення

```
public bool UpdateOrder(int orderId, int clientId, int userId, DateTime orderDate, DateTime?
dueDate, string status, decimal totalAmount)
{
    using (var connection = new MySqlConnection(connectionString))
    {
        string query = @"
            UPDATE Orders
            SET
                ClientID = @ClientID,
                UserID = @UserID,
                OrderDate = @OrderDate,
                DueDate = @DueDate,
                Status = @Status,
                TotalAmount = @TotalAmount
            WHERE OrderID = @OrderID";

        using (var command = new MySqlCommand(query, connection))
        {
            command.Parameters.AddWithValue("@ClientID", clientId);
            command.Parameters.AddWithValue("@UserID", userId);
            command.Parameters.AddWithValue("@OrderDate", orderDate);
            command.Parameters.AddWithValue("@DueDate", dueDate.HasValue ? dueDate.Value :
(object)DBNull.Value);
            command.Parameters.AddWithValue("@Status", status);
            command.Parameters.AddWithValue("@TotalAmount", totalAmount);
            command.Parameters.AddWithValue("@OrderID", orderId);

            connection.Open();
            int rowsAffected = command.ExecuteNonQuery();
            return rowsAffected > 0; // true, якщо замовлення оновлено
        }
    }
}
```

Видалення замовлення

```
public bool DeleteOrder(int orderId)
{
    using (var connection = new MySqlConnection(connectionString))
    {
        string query = "DELETE FROM Orders WHERE OrderID = @OrderID";

        using (var command = new MySqlCommand(query, connection))
```

```
        {
            command.Parameters.AddWithValue("@OrderID", orderId);
            connection.Open();
            int rowsAffected = command.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
}
```

Отримання всіх замовлень для менеджера

```
public List<OrderModel> GetAllOrdersForManager(int managerId)
{
    var orders = new List<OrderModel>();

    using (var connection = new MySqlConnection(connectionString))
    {
        string query = @"
            SELECT
                o.OrderID,
                c.Name AS ClientName,
                o.Status,
                o.OrderDate,
                o.DueDate,
                o.TotalAmount
            FROM Orders o
            INNER JOIN Clients c ON o.ClientID = c.ClientID
            WHERE o.UserID = @ManagerId
            ORDER BY o.OrderDate DESC";

        using (var command = new MySqlCommand(query, connection))
        {
            command.Parameters.AddWithValue("@ManagerId", managerId);
            connection.Open();
            using (var reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    var order = new OrderModel
                    {
                        OrderID = reader.GetInt32("OrderID"),
                        ClientName = reader.GetString("ClientName"),
                        Status = reader.GetString("Status"),
                        OrderDate = reader.GetDateTime("OrderDate"),
                        DueDate = reader.IsDBNull(reader.GetOrdinal("DueDate"))
                                  ? (DateTime?)null
                                  : reader.GetDateTime("DueDate"),
                        TotalAmount = reader.GetDecimal("TotalAmount")
                    };
                    orders.Add(order);
                }
            }
        }
    }

    return orders;
}
```

Фільтрація замовлень за датою та статусом виконання

```
public List<OrderModel> GetFilteredOrders(int managerId, DateTime? startDate, DateTime? endDate,
string status)
{
    var orders = new List<OrderModel>();

    using (var connection = new MySqlConnection(connectionString))
    {
        // Початок SQL-запиту
        string query = @"
            SELECT
                o.OrderID,
                c.Name AS ClientName,
                o.Status,
                o.OrderDate,
                o.DueDate,
                o.TotalAmount
            FROM Orders o
            INNER JOIN Clients c ON o.ClientID = c.ClientID
            WHERE o.UserID = @ManagerId";

        // Додаткові умови фільтрації
        if (startDate.HasValue)
            query += " AND o.OrderDate >= @StartDate";

        if (endDate.HasValue)
            query += " AND o.OrderDate <= @EndDate";

        if (!string.IsNullOrEmpty(status) && status != "Всі")
            query += " AND o.Status = @Status";

        query += " ORDER BY o.OrderDate DESC";

        using (var command = new MySqlCommand(query, connection))
        {
            command.Parameters.AddWithValue("@ManagerId", managerId);

            if (startDate.HasValue)
                command.Parameters.AddWithValue("@StartDate", startDate.Value.Date);

            if (endDate.HasValue)
                command.Parameters.AddWithValue("@EndDate", endDate.Value.Date);

            if (!string.IsNullOrEmpty(status) && status != "Всі")
                command.Parameters.AddWithValue("@Status", status);

            connection.Open();
            using (var reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    var order = new OrderModel
                    {
                        OrderID = reader.GetInt32("OrderID"),
                        ClientName = reader.GetString("ClientName"),
                        Status = reader.GetString("Status"),
```

```
                        OrderDate = reader.GetDateTime("OrderDate"),
                        DueDate = reader.IsDBNull(reader.GetOrdinal("DueDate"))
                                  ? (DateTime?)null
                                  : reader.GetDateTime("DueDate"),
                        TotalAmount = reader.GetDecimal("TotalAmount")
                    };
                    orders.Add(order);
                }
            }
        }
    }

    return orders;
}
```

Отримання всіх поточних запасів на складі для комірника

```
public List<StockItemModel> GetCurrentStockItems()
{
    var stockItems = new List<StockItemModel>();

    using (var connection = new MySqlConnection(connectionString))
    {
        string query = @"
            SELECT
                ItemID,
                Name,
                Type,
                Unit,
                Quantity,
                MinThreshold
            FROM StockItems
            ORDER BY Name ASC";

        using (var command = new MySqlCommand(query, connection))
        {
            connection.Open();
            using (var reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    var item = new StockItemModel
                    {
                        ItemID = reader.GetInt32("ItemID"),
                        Name = reader.GetString("Name"),
                        Type = reader.GetString("Type"), // 'Сировина' або 'Продукція'
                        Unit = reader.GetString("Unit"),
                        Quantity = reader.GetDecimal("Quantity"),
                        MinThreshold = reader.GetDecimal("MinThreshold")
                    };
                    stockItems.Add(item);
                }
            }
        }
    }
    return stockItems;
}
```