НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____Ігор БОЛБОТ	**_____Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____2026 р.	«___» _____2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення системи обліку послуг станції технічного обслуговування»
Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Юрій МІЛОВІДОВ**
(підпис)
ст.викладач

Виконав _____ **Владислав ПІДДУБНИЙ**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ

«_19_» ___грудня_____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Піддубному Владиславу Олеговичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення системи обліку послуг станції технічного обслуговування»

затверджена наказом від «__19_» ____ грудня ___ 2025 р. № ___3204__ «С»

Термін подання завершеної роботи на кафедру «_22_» ___травня____ 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): облік та опис роботи малого бізнесу.

Перелік питань, що підлягають дослідженню:

1. Аналіз предметної області системи обліку технічного обслуговування
2. Проєктування інформаційного та програмного забезпечення
3. Розробка алгоритмічного забезпечення
4. Тестування та оцінювання ефективності системи

Дата видачі завдання «_19__» _____ грудня ____ 2025 р.

Керівник бакалаврської кваліфікаційної робота _____ **Юрій МІЛОВІДОВ**
(підпис)

Завдання взяв до виконання _____ **Владислав ПІДДУБНИЙ**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

– API - Application Programming Interface, програмний інтерфейс взаємодії між компонентами системи

– CSV - Comma-Separated Values, текстовий формат для експорту табличних даних

– CRUD - Create, Read, Update, Delete, базові операції створення, перегляду, оновлення та видалення даних

– FK - Foreign Key, зовнішній ключ таблиці бази даних

– GUI - Graphical User Interface, графічний інтерфейс користувача

– ID - унікальний ідентифікатор запису в базі даних

– KPI - Key Performance Indicators, ключові показники ефективності

– PK - Primary Key, первинний ключ таблиці бази даних

– QSS - Qt Style Sheets, засіб стилізації інтерфейсу у PyQt6

– SQL - Structured Query Language, мова структурованих запитів до бази даних

– SQLite - вбудована реляційна система керування базами даних

– UI - User Interface, інтерфейс користувача

– UX - User Experience, користувацький досвід

– VIN - Vehicle Identification Number, ідентифікаційний номер транспортного засобу

– БД - база даних

– ІС - інформаційна система

– ОС - операційна система

– ПЗ - програмне забезпечення

– СКБД - система керування базами даних

– СТО - станція технічного обслуговування

– CSV-звіт - файл звіту у форматі CSV для подальшого перегляду або оброблення

ВСТУП

Сучасні станції технічного обслуговування (СТО) працюють у висококонкурентному середовищі, де ефективність бізнес-процесів безпосередньо визначає якість обслуговування клієнтів, швидкість виконання робіт та економічні показники підприємства. Традиційні моделі ведення обліку - ручне фіксування замовлень, паперові журнали, операції в окремих незв’язаних програмних модулях - не відповідають потребам цифрової економіки, оскільки створюють ризики затримок, дублювання даних, помилок персоналу та втрати аналітичної інформації. Зростання кількості транспортних засобів, підвищення вимог до точності технічної діагностики та необхідність оперативного управління запасами формують запит на комплексні інформаційні системи обліку, здатні автоматизувати всі етапи роботи СТО - від реєстрації клієнта до формування детального звіту про виконані операції. Саме тому актуальною є розробка програмного забезпечення облікової системи, яке забезпечить інтеграцію довідників, замовлень, складу, майстрів, технологічних карт, фінансових операцій і аналітичних модулів у єдиному інформаційному середовищі, підвищивши прозорість та керованість процесів [1].

Метою роботи є створення комплексного програмного забезпечення облікової системи для станції технічного обслуговування, здатного забезпечити централізоване керування операціями, достовірний облік ресурсів, підтримку прийняття рішень та підвищення ефективності роботи персоналу.

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

1. проаналізувати предметну область, існуючі інформаційні системи та тенденції автоматизації обліку на СТО;
2. сформувати вимоги до функціональних, нефункціональних, технічних і безпекових характеристик системи;
3. розробити UML-моделі, логічну та фізичну структури даних;

4. спроєктувати архітектуру програмного забезпечення та механізми взаємодії компонентів;

5. реалізувати ключові модулі системи обліку замовлень, клієнтів, складу, прайс-листів і звітності;

6. провести тестування та оцінити ефективність роботи системи в умовах типових сценаріїв СТО;

7. сформувати висновки щодо результатів проєктування та перспектив розвитку системи.

Об’єктом дослідження є процеси обліку, планування, адміністрування та контролю робіт на станції технічного обслуговування.

Предметом дослідження є методи, програмні моделі, алгоритмічні засоби та інформаційні технології автоматизації облікових процесів СТО, включно з механізмами оброблення замовлень, управління запасами, контролю виконання робіт і формування звітності.

Методи дослідження включають системний аналіз, структурне й об’єктно-орієнтоване моделювання, UML-проєктування, методи реляційного та документного моделювання баз даних, засоби програмної інженерії, методи оцінювання ефективності програмних систем та елементи аналітичної обробки даних [2].

Наукова новизна роботи полягає у формуванні цілісної концепції інтегрованої облікової системи для СТО, яка забезпечує автоматизоване ведення технічних і бізнес-процесів, включно з управлінням складським обліком, плануванням робіт, фіксацією історії обслуговувань, контролем ресурсів та аналітичним оцінюванням показників діяльності. Запропоновані моделі даних і архітектура системи підвищують рівень узгодженості інформаційних потоків та забезпечують можливість масштабування під різні формати підприємств.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ СИСТЕМИ ОБЛІКУ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ

1.1 Опис предметної області

Предметна область облікової системи для станції технічного обслуговування формується сукупністю бізнес-процесів, пов’язаних із реєстрацією клієнтів і транспортних засобів, обробленням замовлень-нарядів, плануванням та виконанням ремонтних робіт, управлінням запасами та контролем використанням запасних частин, фінансовими операціями, а також аналітикою ефективності роботи СТО. У взаємодії цих процесів виникають складні інформаційні потоки, що охоплюють дані про клієнтів, історію обслуговування, стан ремонту, використані матеріали, фінансові транзакції, показники завантаженості та ключові метрики сервісної діяльності. На рис. 1.1 подано узагальнену концептуальну схему предметної області, яка відображає ключові актори та інформаційні зв’язки між підсистемами.

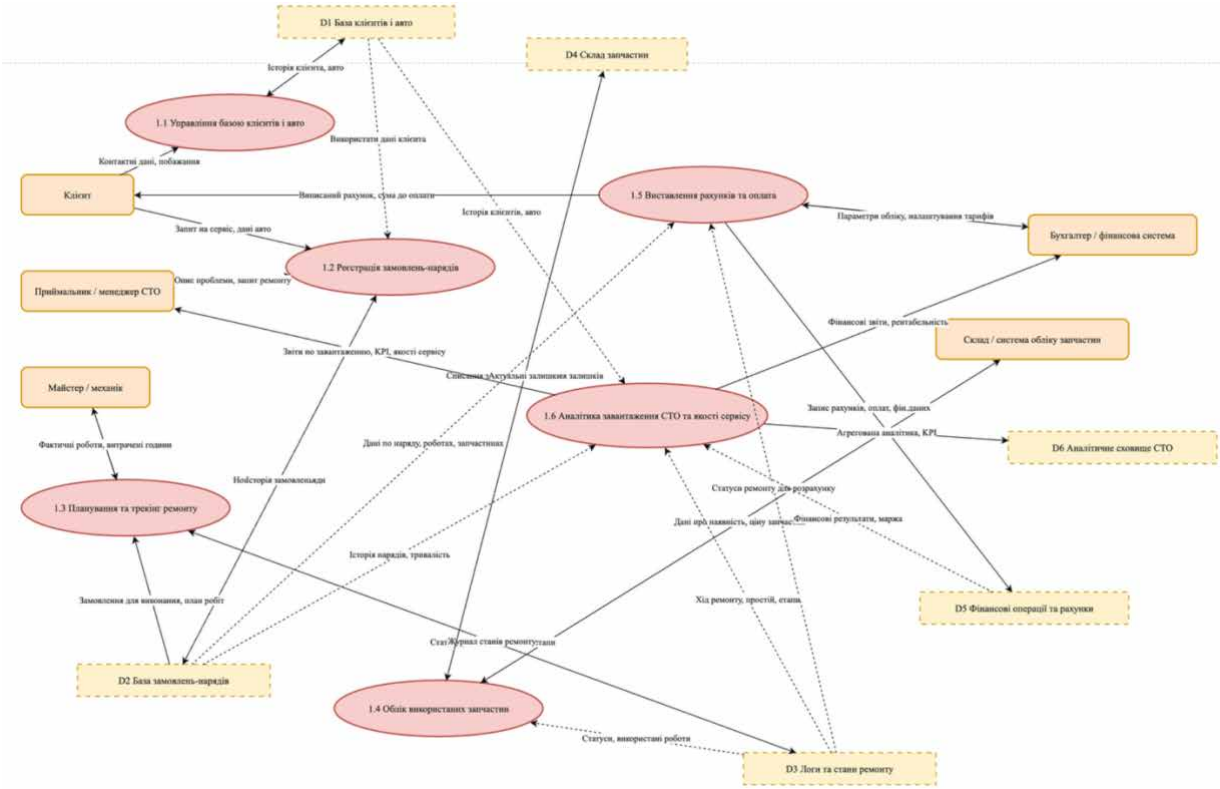

Рис. 1.1. Концептуальна модель предметної області облікової системи СТО

Згідно з рис. 1.1, домінуючими інформаційними сутностями є клієнт, транспортний засіб, замовлення-наряд, перелік робіт і послуг, складські залишки та фінансові операції, які формують основу для автоматизованого управління сервісним циклом. Облікова система повинна забезпечувати узгоджений обмін даними між процесами реєстрації замовлення, планування робіт, відстеження прогресу ремонту, списання запасних частин і формування рахунків, мінімізуючи ризики розбіжностей та дублювання інформації. Для відображення базових структур даних, типових для діяльності СТО, у табл. 1.1 наведено ключові категорії інформації, що мають бути підтримані системою.

Таблиця 1.1

Основні інформаційні сутності предметної області СТО

Сутність	Опис
Клієнт	Контактні дані, історія звернень, уподобання, пов’язані транспортні засоби
Автомобіль	Марка, модель, рік випуску, VIN, технічні характеристики, історія обслуговувань
Замовлення-наряд	Перелік робіт, дефекти, статус виконання, призначені виконавці, витрати часу
Роботи та послуги	Каталог сервісних операцій, нормативи часу, ціни
Запасні частини	Номенклатура, складські залишки, рух матеріалів, списання
Майстри/механіки	Завантаженість, кваліфікація, фактично виконані роботи
Фінансові дані	Рахунки, оплати, калькуляції, рентабельність
Журнали подій	Логи змін у замовленнях, стан ремонту, історія дій персоналу

Наведені структури забезпечують можливість формування єдиної цифрової екосистеми, у межах якої відбуваються реєстрація клієнта, створення та опрацювання замовлення-наряду, управління запасами, контроль виконання робіт і формування фінансової звітності. Інтеграція цих типів даних забезпечує цілісність інформаційних потоків, підвищує точність обліку та створює умови для впровадження аналітичних методів оцінювання завантаженості СТО, якості сервісу та ефективності ремонтних процесів. З огляду на зростання обсягів

операційної інформації та підвищені вимоги до швидкості прийняття рішень, предметна область вимагає застосування сучасних засобів інформаційного моделювання та програмної інженерії [1].

1.2 Аналіз існуючих рішень й інформаційних систем

Сучасний ринок програмних продуктів для станцій технічного обслуговування представлений широким спектром рішень, орієнтованих на автоматизацію роботи з клієнтами, управління замовленнями, складським обліком та фінансовими операціями. Більшість платформ реалізує модель наскрізного цифрового супроводу сервісного циклу — від формування попередньої заявки до закриття рахунку та аналітичної оцінки ефективності діяльності СТО. На рис. 1.2–1.6 наведено приклади інтерфейсів популярних комерційних систем, які формують технологічну основу сучасних галузевих рішень.

На рис. 1.2 подано робочу панель системи AutoLeap, що реалізує канбан-модель управління замовленнями. Платформа забезпечує реєстрацію клієнтів, ведення каталогу послуг, управління запасними частинами, контроль статусів ремонтів і фінансову аналітику, що дозволяє підвищити прозорість операцій та оптимізувати планування.

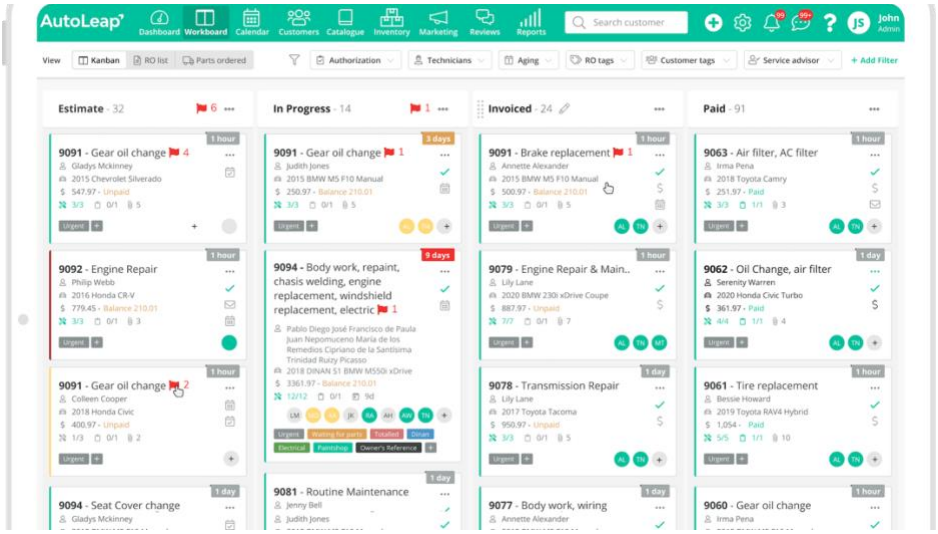

Рис. 1.2. Інтерфейс системи AutoLeap із модулем управління сервісними замовленнями

На рис. 1.3 наведено систему Shopmonkey, яка підтримує повний сервісний цикл СТО та має розвинені механізми планування робіт і комунікації з клієнтами. Характерною рисою є візуально насичений канбан-простір, інтеграція з фінансовими модулями та підтримка інвентаризації.

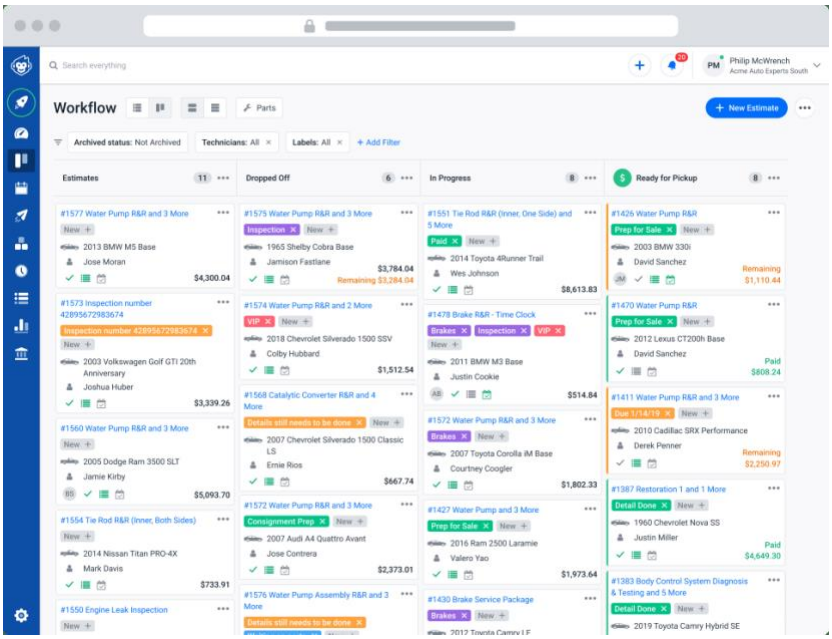

Рис. 1.3. Робочий процес управління замовленнями у системі Shopmonkey

На рис. 1.4 зображено систему Tekmetric, яка робить акцент на структурованих бізнес-процесах, швидкому створенні замовлень-нарядів, відстеженні статусів та формуванні звітів щодо рентабельності робіт. Платформа орієнтована на мультифіліальні СТО з підвищеними вимогами до контролю доступу та журналювання.

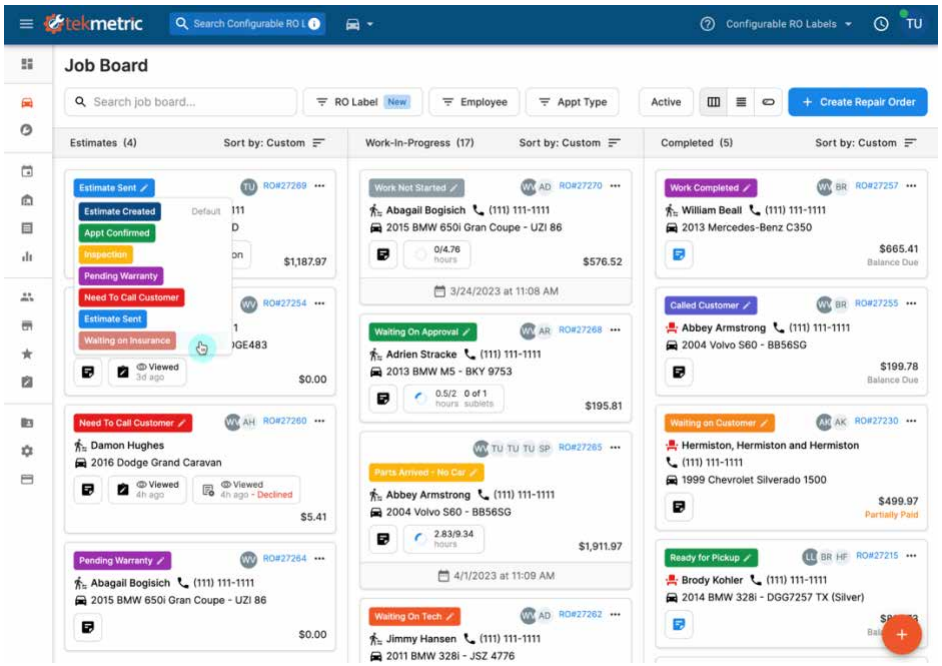

Рис. 1.4. Інтерфейс Tekmetric для обліку замовлень-нарядів

На рис. 1.5 продемонстровано модуль обліку запасів системи Klipboard, який забезпечує роботу з номенклатурою, залишками, рухом матеріалів, маржинальністю та налаштуваннями інвентаризації. Наявність розширених каталогів постачальників і сервісних контрактів робить систему придатною для застосування в складних сервісних середовищах.

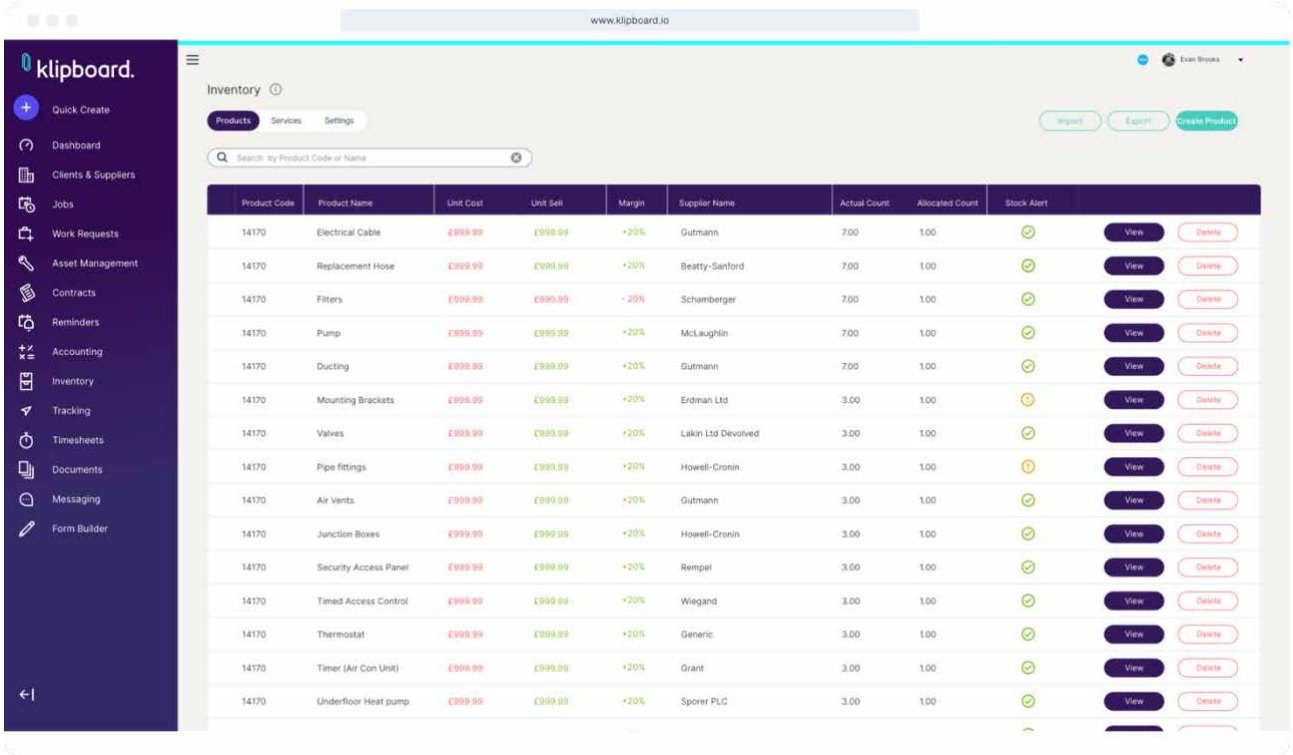

Рис. 1.5. Модуль складського обліку системи Klipboard

На рис. 1.6 наведено приклад аналітичної панелі комплексної сервісної системи, орієнтованої на бюджетування, моніторинг витрат, оцінювання прибутковості й динаміки виконання замовлень. Подібні панелі дозволяють оперативно контролювати фінансовий стан СТО, оцінювати завантаженість майстрів і відстежувати ключові показники ефективності.

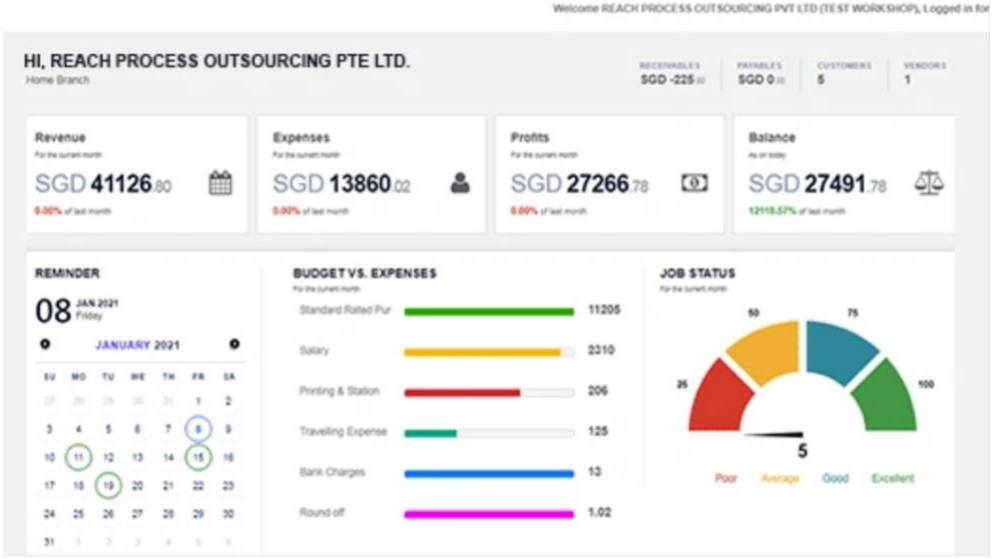

Рис. 1.6. Аналітична панель оцінювання фінансових і сервісних показників

Для систематизації особливостей розглянутих рішень у табл. 1.2 наведено порівняльну характеристику функціональних можливостей, де додатково включено проектовану систему СТО, орієнтовану на інтегрований облік замовлень-нарядів, керування складом, контроль завантаженості майстрів та формування аналітики якості сервісу.

Таблиця 1.2

Порівняльна характеристика існуючих рішень та розроблюваної системи

Система	Управління замовленнями	Облік клієнтів та авто	Склад та запчастини	Планування робіт	Фінансові модулі	Аналітика KPI	Особливості
AutoLeap	✓	✓	✓	✓	✓	✓	Акцент на цифрових процесах і CRM
Shopmonkey	✓	✓	✓	✓	✓	✓	Розширений канбан,

							комунікація з клієнтом

Продовження таблиці 1.2

Tekmetric	✓	✓	✓	✓	✓	✓	Орієнтація на прозорість сервісних процесів
Klipboard	Частково	✓	✓	Частково	✓	✓	Сильний складський модуль і контракти
Dashboard-аналітика	Частково	Частково	Частково	Частково	✓	✓	Поглиблена фінансова аналітика
Розроблювана система СТО	✓	✓	✓ (склад + авто-запчастини)	✓ (гнучке планування та трекінг)	✓ (рахунки, оплата, калькуляції)	✓ (KPI, завантаженість, якість сервісу)	Єдина інтегрована система з наскрізним обліком, оптимізацією процесів і підтримкою аналітики

Аналіз існуючих продуктів свідчить, що більшість із них забезпечує базові функції СТО, однак не всі підтримують глибоку інтеграцію процесів, наскрізну аналітику та можливість адаптації до специфічних моделей роботи сервісного центру. Проєктована система покликана усунути ці обмеження шляхом створення єдиної інформаційної платформи, що охоплює управління клієнтами, замовленнями, запасами, плануванням робіт і аналітикою сервісних операцій, забезпечуючи цілісність, масштабованість і підвищену керованість операційної діяльності СТО.

1.3 Моделювання предметної області

Моделювання предметної області облікової системи для СТО є критично важливим етапом проєктування, оскільки забезпечує формалізацію ролей,

процесів, інформаційних потоків і взаємодій між користувачами та функціональними модулями системи. Побудова UML-моделей дає змогу структурувати бізнес-логіку сервісного центру, визначити межі відповідальності акторів, ідентифікувати ключові прецеденти, а також описати часову й операційну послідовність виконання ремонтних робіт. На рис. 1.7–1.9 наведено базові UML-діаграми, що відображають предметну область у форматі прецедентів, послідовностей та бізнес-процесів.

На рис. 1.7 подано діаграму прецедентів, що описує основні функції CRM-системи СТО, включно з реєстрацією клієнтів, створенням замовлень-нарядів, плануванням та призначенням робіт, трекінгом стану ремонту, обліком запасних частин, формуванням рахунку, управлінням правами доступу та аналітикою завантаженості й якості сервісу. Діаграма відображає взаємодію між ключовими акторами - клієнтом, приймальником/менеджером СТО, майстром, бухгалтером та адміністратором системи - що визначає функціональні межі подальшої реалізації.

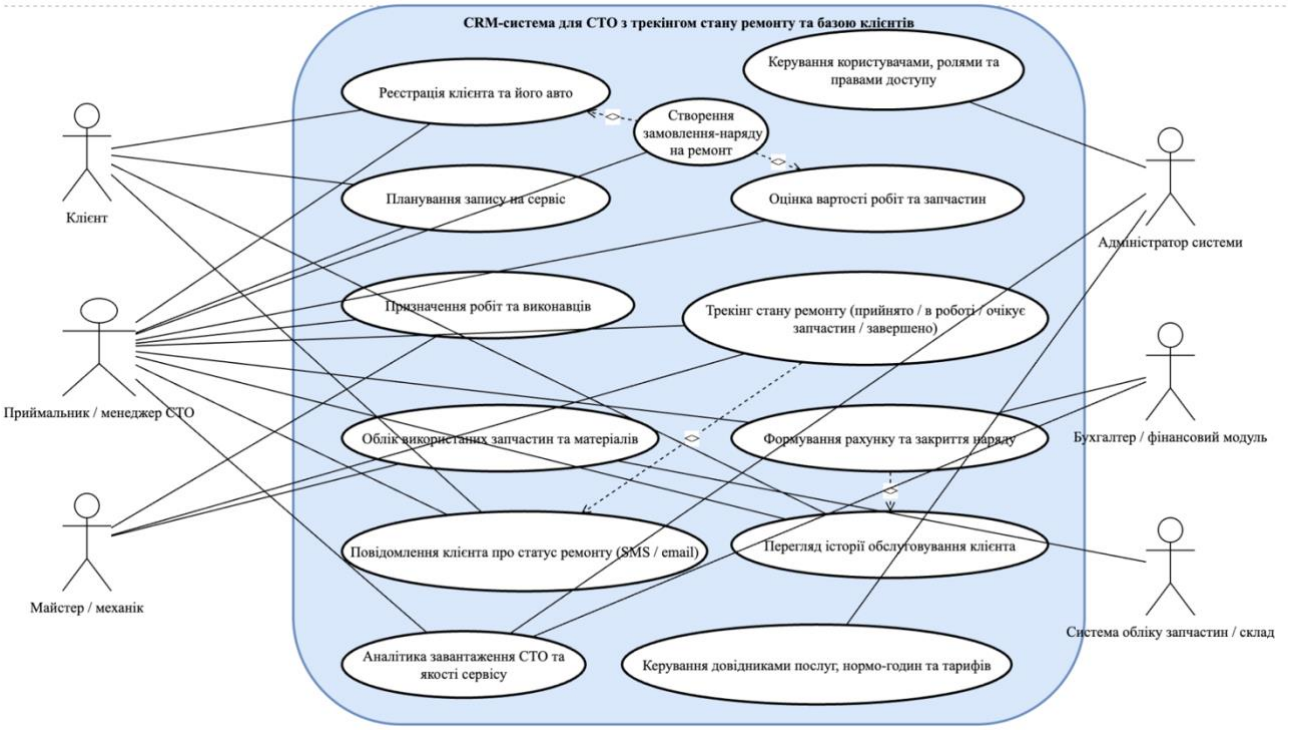

Рис. 1.7 – Діаграма прецедентів предметної області CRM-системи СТО

На рис. 1.8 наведено діаграму послідовності процесу оброблення звернення клієнта - від запису на сервіс і створення наряду до формування

кошторису, підтвердження клієнтом, оновлення статусу ремонту й надсилання повідомлень. Діаграма демонструє послідовність взаємодії між менеджером, CRM-інтерфейсом, модулем замовлень-нарядів, складом та модулем нотифікацій, що дозволяє формалізувати часові обмеження (t0–t1, t1–t2) і критичні точки процесу.

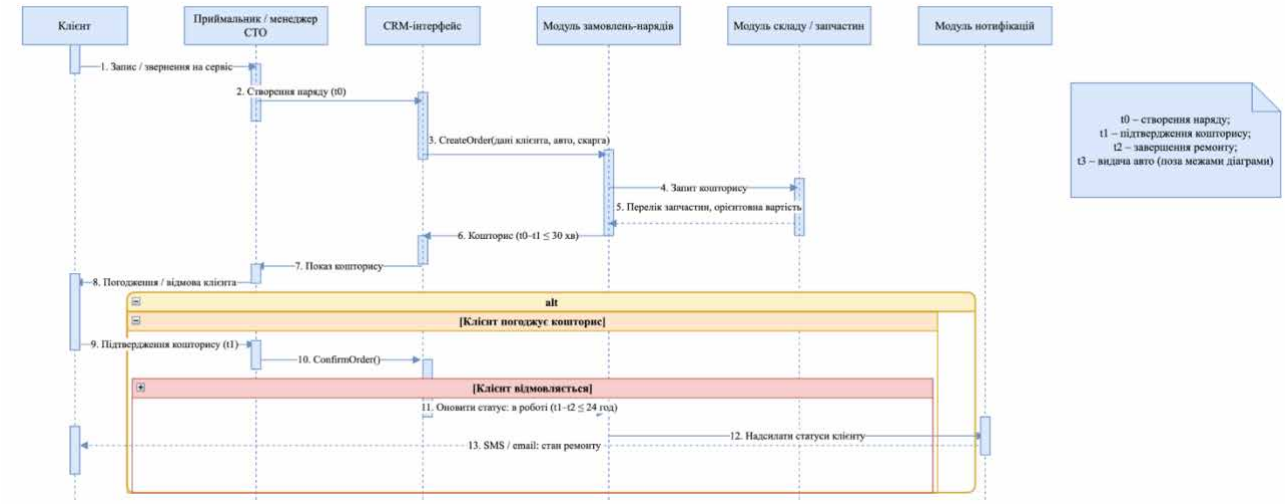

Рис. 1.8 – Діаграма послідовності процесу створення та погодження замовлення-наряду

На рис. 1.9 зображено діаграму активності, яка формалізує бізнес-процес оброблення звернення клієнта: прийняття заявки, реєстрація клієнта та автомобіля, формування попереднього кошторису, прийняття рішення клієнтом, створення або скасування замовлення-наряду, оновлення статусу ремонту та надсилання нотифікацій. Модель відображає рольові межі між клієнтом, менеджером та системою, що забезпечує чітке розділення відповідальності та послідовності операцій.

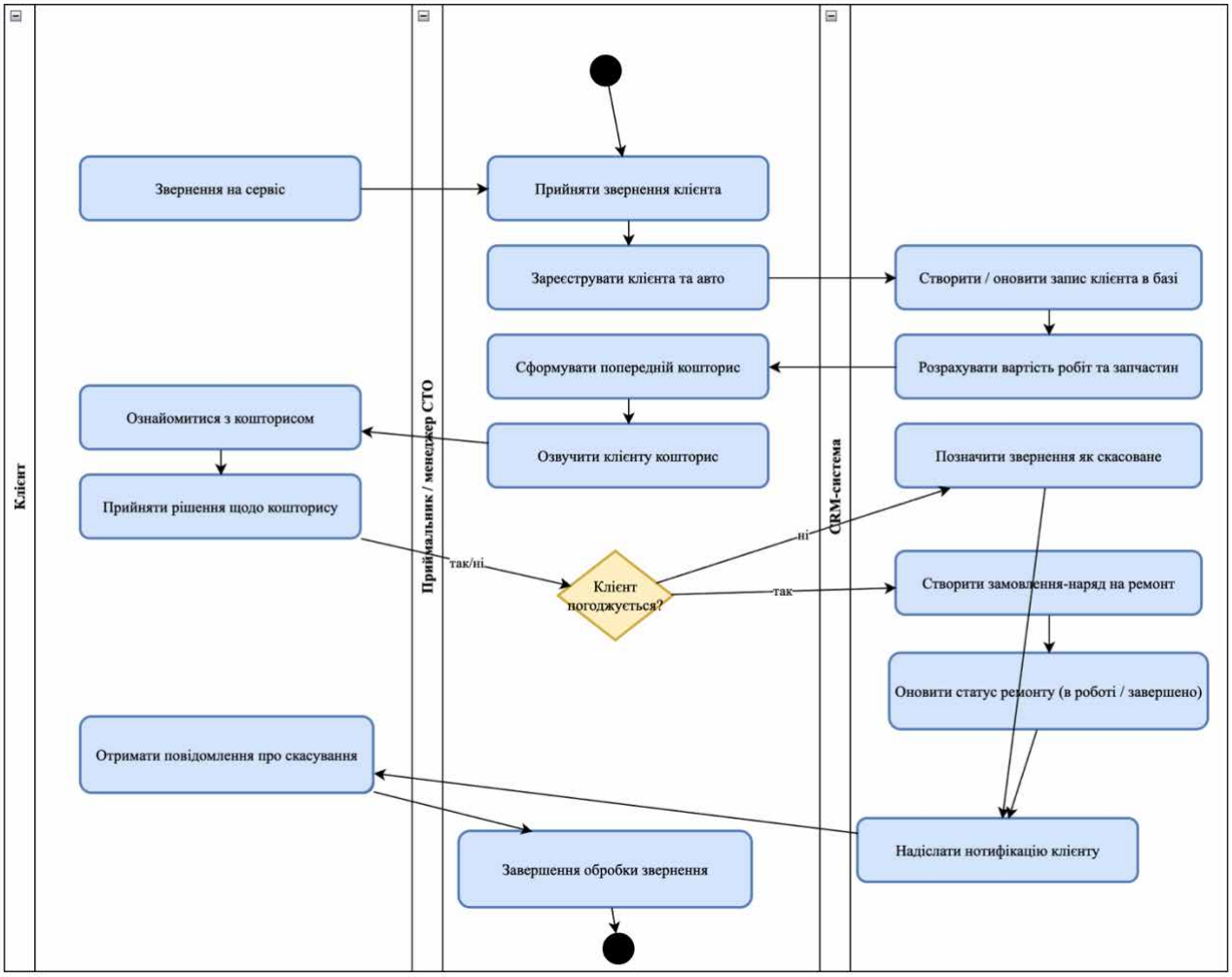

Рис. 1.9 – Діаграма активності бізнес-процесу оброблення звернення клієнта

Моделювання предметної області дозволяє створити формалізоване уявлення про ключові бізнес-процеси СТО та забезпечує підґрунтя для побудови логічної структури даних, визначення функціональних вимог і проєктування архітектури програмного забезпечення. Усі наведені UML-моделі описують типовий сервісний цикл роботи станції технічного обслуговування й забезпечують узгодженість подальших етапів проєктування системи.

1.4 Аналіз вимог інформаційної системи

Аналіз вимог є ключовим етапом формування архітектури програмного забезпечення, оскільки він визначає функціональні можливості, обмеження,

умови експлуатації та критерії якості майбутньої облікової системи для СТО. Вимоги структуровано за категоріями: функціональні, нефункціональні та вимоги до безпеки. На основі типових сценаріїв роботи сервісного центру, моделювання предметної області (розділи 1.1–1.3) та аналізу існуючих систем (розділ 1.2) сформовано узагальнені таблиці вимог. У табл. 1.3 подано функціональні вимоги, що відображають основні операції системи: управління клієнтами, створення замовлень-нарядів, трекінг статусів ремонту, облік запчастин, формування рахунків та аналітика завантаження СТО.

Таблиця 1.3

Функціональні вимоги облікової системи СТО

№	Функціональна вимога	Опис
F1	Реєстрація клієнтів та авто	Зберігання контактів, історії обслуговувань, технічних параметрів авто
F2	Створення замовлення-наряду	Фіксація скарги, опис проблем, вибір робіт та матеріалів
F3	Планування робіт	Призначення виконавців, формування графіку, контроль завантаження
F4	Трекінг ремонту	Статуси ремонту: прийнято, в роботі, очікує запчастини, завершено
F5	Облік запчастин	Списання, залишки, рух матеріалів, перевірка наявності
F6	Формування рахунків	Автоматичні калькуляції вартості робіт та матеріалів
F7	Нотифікації клієнтів	Надсилання SMS/email про статус ремонту
F8	Аналітика завантаженості	KPI, рентабельність робіт, звіти по майстрах, часах та матеріалах
F9	Управління правами доступу	Ролі: адміністратор, менеджер, майстер, бухгалтер

Функціональні вимоги (табл. 1.3) окреслюють обов’язкові модулі та очікувану поведінку системи, орієнтуючи подальше проєктування логічної та фізичної моделі даних.

Нефункціональні вимоги визначають рівень продуктивності, доступності, масштабованості та ергономічності системи, що є критично важливим у

контексті щоденної роботи СТО. На основі моделювання процесів та операційних потреб сформовано табл. 1.4.

Таблиця 1.4

Нефункціональні вимоги облікової системи СТО

№	Нефункціональна вимога	Значення / критерій
NF1	Максимальний час відповіді інтерфейсу	≤ 200 мс для основних операцій
NF2	Продуктивність обробки замовлення	≤ 30 с для формування кошторису
NF3	Доступність системи	≥ 99.5% часу роботи
NF4	Масштабованість	Підтримка 10–50 одночасних користувачів
NF5	Ергономічність	Інтерфейс не більше 3 кліків до ключових функцій
NF6	Надійність збереження даних	ACID-гарантії для транзакцій БД
NF7	Інтеграція	Можливість зв’язку зі складом, SMS/email сервісами

Таблиця 1.4 визначає цільові метрики ефективності роботи системи, необхідні для стабільного функціонування в умовах інтенсивного сервісного навантаження.

Важливою частиною проєктування є вимоги до безпеки, які забезпечують захист даних клієнтів, фінансової інформації, історії ремонту, доступів працівників та контроль операцій у системі. Узагальнені вимоги подано в табл. 1.5.

Таблиця 1.5

Вимоги до безпеки облікової системи СТО

№	Вимога до безпеки	Опис
S1	Аутентифікація користувачів	Логін/пароль, сесійний токен
S2	Рольова модель доступу (RBAC)	Розмежування прав між менеджером, майстром, бухгалтером, адміністратором
S3	Шифрування даних	TLS 1.3 при передачі, шифрування критичних полів у БД

Продовження таблиці 1.5

S4	Логи дій користувачів	Фіксація створення/редагування замовлень, змін статусів і даних
S5	Контроль доступу до фінансових операцій	Обмеження для груп користувачів, окремі повноваження на закриття наряду
S6	Захист від несанкціонованих змін	Перевірки цілісності, аудит змін
S7	Резервне копіювання	Щоденний бекап БД, аварійне відновлення

Сформовані функціональні, нефункціональні та безпекові вимоги створюють комплексну основу для подальшого проєктування архітектури системи, логічних моделей даних і програмних компонентів. Вони враховують особливості сервісних процесів СТО, типові операційні сценарії та вимоги до швидкодії, надійності й захищеності інформації, що гарантує коректність і узгодженість наступних етапів розроблення.

1.5 Постановка завдання

Постановка завдання визначає цілісну структуру функціонування облікової системи для станції технічного обслуговування, окреслює межі її застосування та встановлює відповідність між вхідними даними, обчислювальними процесами та вихідною інформацією, яку система повинна генерувати. На основі аналізу предметної області, моделювання процесів (рис. 1.7–1.9) та сформованих функціональних і нефункціональних вимог (табл. 1.3–1.5) узагальнено базові характеристики інформації, що надходить у систему та формується нею у відповідь. У табл. 1.6 подано структуру вхідних і вихідних даних, які забезпечують реалізацію сервісного циклу від моменту звернення клієнта до формування рахунку і завершення робіт.

Таблиця 1.6

Вхідні та вихідні дані облікової системи СТО

Тип даних	Зміст
Вхідні дані	Дані клієнта та авто; опис звернення; перелік скарг; параметри первинного огляду; інформація про запчастини та їх наявність; трудові нормативи; дані про фактично виконані роботи та витрачені матеріали; фінансові параметри (ціни, знижки, тарифи); статуси ремонту; дії користувачів системи
Вихідні дані	Створений замовлення-наряд; оновлені дані клієнта та історії обслуговування; кошторис; статуси етапів ремонту; інформація про використані запчастини; сформований рахунок; фінансові звіти; аналітичні показники завантаженості СТО та KPI; повідомлення клієнту; журнали подій

Завдання розроблення системи полягає у створенні програмного забезпечення, здатного забезпечити повний цикл автоматизації сервісних процесів СТО: реєстрацію клієнтів та їх транспортних засобів, формування замовлень-нарядів, призначення робіт і виконавців, облік запасних частин, трекінг стану ремонту, формування фінансових документів, надсилання нотифікацій та аналітичне оцінювання ефективності роботи. Система повинна функціонувати як єдина інформаційна платформа з централізованим сховищем даних, ролями доступу та механізмами забезпечення цілісності інформації. Постановка завдання передбачає реалізацію механізмів інтеграції з модулем складського обліку та сервісами нотифікацій, забезпечення високої продуктивності інтерфейсу при інтенсивних операціях, а також дотримання вимог безпеки щодо захисту персональних і фінансових даних. Створення такої системи спрямоване на підвищення прозорості сервісних процесів, зменшення операційних витрат, оптимізацію завантаженості майстрів і забезпечення клієнтів високим рівнем сервісу за рахунок достовірної інформаційної підтримки всіх етапів ремонту.

1.6 Висновки до першого розділу

У першому розділі здійснено системний аналіз предметної області облікової системи станції технічного обслуговування, що дозволило сформувати цілісне уявлення про структуру бізнес-процесів, інформаційні потоки та функціональні взаємозв’язки між учасниками сервісного циклу. На основі розгляду специфіки роботи СТО визначено ключові сутності доменної моделі, серед яких клієнт, транспортний засіб, замовлення-наряд, процеси планування та виконання ремонту, облік запасних частин, фінансові операції та аналітичні показники сервісної діяльності. Проведений аналіз підтверджує необхідність використання інтегрованого програмного забезпечення, здатного забезпечити цілісність даних і забезпечити підтримку всіх етапів обслуговування.

Огляд існуючих програмних рішень демонструє тенденцію до побудови комплексних CRM/ERP-платформ, орієнтованих на автоматизацію взаємодії з клієнтом, оптимізацію робіт майстрів, управління складом та формування детальної аналітики. Разом із тим, представлені системи нерідко мають обмеження у гнучкості адаптації, глибині інтеграції з внутрішніми процесами СТО та можливостях точкового налаштування під специфічні бізнес-моделі. Це обґрунтовує доцільність розроблення власної облікової системи, що враховує потреби підприємства та забезпечує більш тісну інтеграцію процесів.

Моделювання предметної області за допомогою UML-діаграм - прецедентів, послідовності та активності - дозволило формалізувати ролі користувачів, їхні взаємодії з системою, часову логіку виконання сервісних операцій і ключові етапи оброблення звернення клієнта. Ці моделі визначають логічну структуру майбутньої системи, сприяють усуненню неоднозначностей та закладають основу для архітектурного проєктування.

У результаті структурованого аналізу вимог сформовано набір функціональних, нефункціональних і безпекових характеристик, що

визначають очікувані властивості системи та критерії її ефективності. Виокремлено вимоги до продуктивності, масштабованості, доступності та захисту персональних і фінансових даних, що є критично важливими у контексті роботи сервісного центру. Визначено основні вхідні та вихідні дані, на підставі яких система повинна функціонувати як надійний інструмент підтримки управлінських та операційних процесів СТО.

Перший розділ формує методологічну та інформаційну базу для подальших етапів розроблення облікової системи - побудови архітектури, проєктування модулів, розроблення бази даних і реалізації програмного забезпечення. Він забезпечує чітке розуміння задачі, визначає вимоги до системи та забезпечує концептуальну узгодженість усіх наступних розділів кваліфікаційної роботи.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Для проєктування інформаційного забезпечення облікової системи СТО було розроблено логічну модель даних у вигляді ER-діаграми, подану на рис. 2.1. Модель відображає основні інформаційні сутності системи, їх атрибути, первинні та зовнішні ключі, а також зв’язки між об’єктами предметної області.

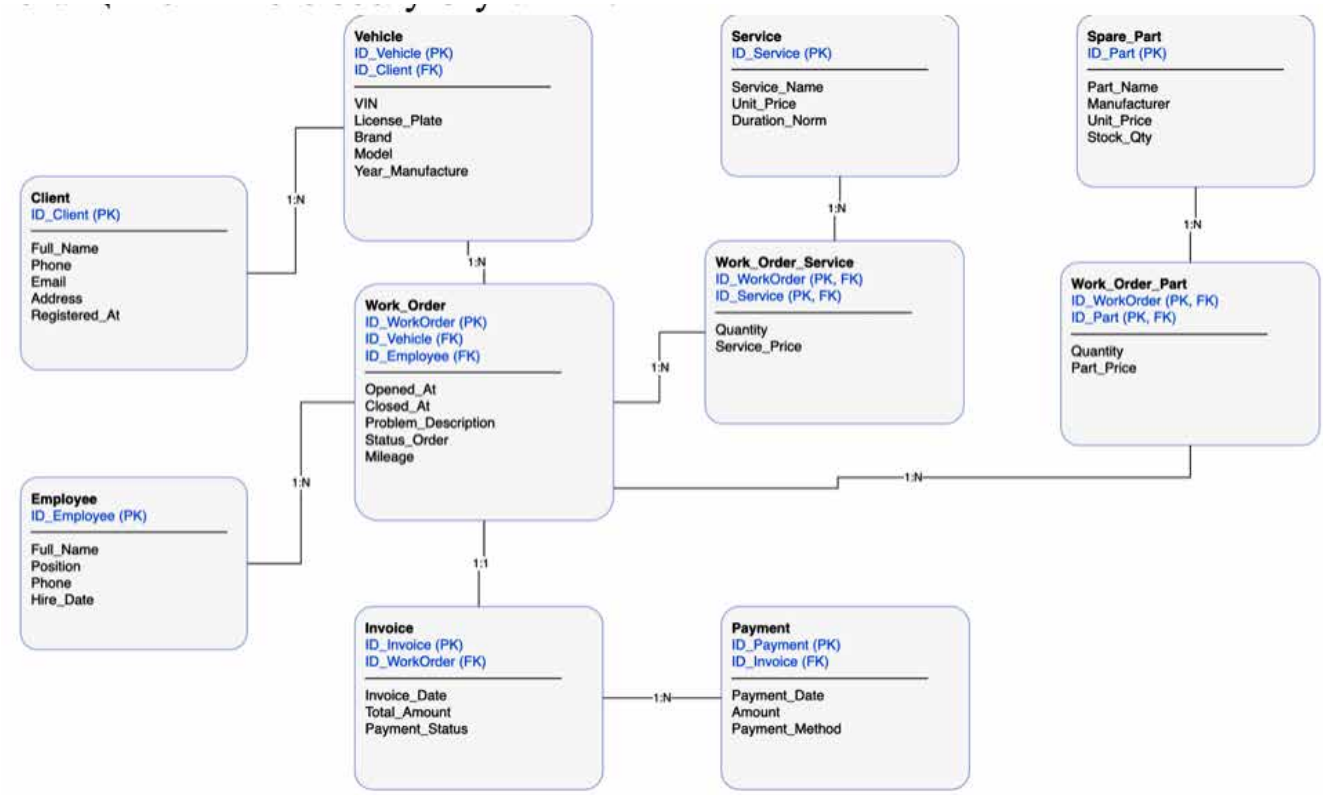

Рис. 2.1. Логічна модель даних облікової системи СТО

Основу логічної моделі становлять сутності Client, Vehicle, Employee, Work_Order, Service, Spare_Part, Invoiceта Payment. Центральною сутністю є Work_Order, оскільки саме замовлення-наряд поєднує дані про автомобіль, відповідального працівника, виконані роботи, використані запчастини, рахунок і оплату.

Зв’язок між клієнтом і автомобілем має тип 1:N, оскільки один клієнт може мати декілька транспортних засобів. Сутність Vehicle пов’язана із

замовленнями-нарядами також зв’язком 1:N, що дозволяє зберігати повну історію обслуговування кожного автомобіля. Сутність Employee пов’язана з Work_Order, оскільки один працівник може бути відповідальним за декілька замовлень.

Для уникнення дублювання даних про послуги та запасні частини у моделі використано проміжні сутності Work_Order_Service та Work_Order_Part. Вони реалізують зв’язки типу M:N між замовленням-нарядом і довідниками послуг та запчастин. Такий підхід дозволяє в межах одного замовлення фіксувати декілька робіт і декілька використаних деталей, зберігаючи при цьому кількість, вартість послуг і ціну списаних запчастин.

Фінансовий блок представлено сутностями Invoice та Payment. Для кожного замовлення-наряду формується один рахунок, тому між Work_Order і Invoice задано зв’язок 1:1. Оплати пов’язані з рахунком зв’язком 1:N, оскільки один рахунок може бути оплачений частинами або різними способами.

Під час побудови логічної моделі було враховано принципи нормалізації даних. Модель приведено щонайменше до третьої нормальної форми. Перша нормальна форма забезпечується тим, що всі атрибути мають атомарні значення, наприклад окремо зберігаються телефон, email, VIN, номерний знак, дата відкриття замовлення та статус. Друга нормальна форма досягається завдяки тому, що неключові атрибути залежать від повного первинного ключа відповідної сутності. Третя нормальна форма забезпечується винесенням довідкових даних про послуги та запчастини в окремі таблиці, що усуває транзитивні залежності та дублювання інформації.

Основні елементи нормалізації логічної моделі наведено в табл. 2.1.

Таблиця 2.1 - Нормалізація логічної моделі даних

Рівень нормалізації	Реалізація в моделі
1НФ	Усі поля мають атомарні значення, повторювані групи відсутні
2НФ	Атрибути кожної таблиці залежать від її первинного ключа
3НФ	Довідники послуг, запчастин, клієнтів і авто винесені в окремі сутності
Усунення дублювання	Послуги та запчастини не дублюються в замовленнях, а підключаються через проміжні таблиці
Підтримка цілісності	Зв’язки між сутностями реалізовані через зовнішні ключі

Запропонована логічна модель забезпечує цілісне зберігання даних про клієнтів, автомобілі, працівників, замовлення-наряди, виконані послуги, використані запчастини, рахунки та платежі. Вона є основою для подальшого створення фізичної моделі бази даних і реалізації програмних модулів облікової системи СТО.

2.2 Представлення діаграми класів і кооперацій системи

Для опису програмної структури облікової системи СТО було розроблено UML-діаграму класів, подану на рис. 2.2. Вона відображає основні об’єкти системи, їх атрибути, операції та зв’язки між класами, які забезпечують реєстрацію клієнтів, облік автомобілів, створення замовлень, виконання послуг і формування рахунків.

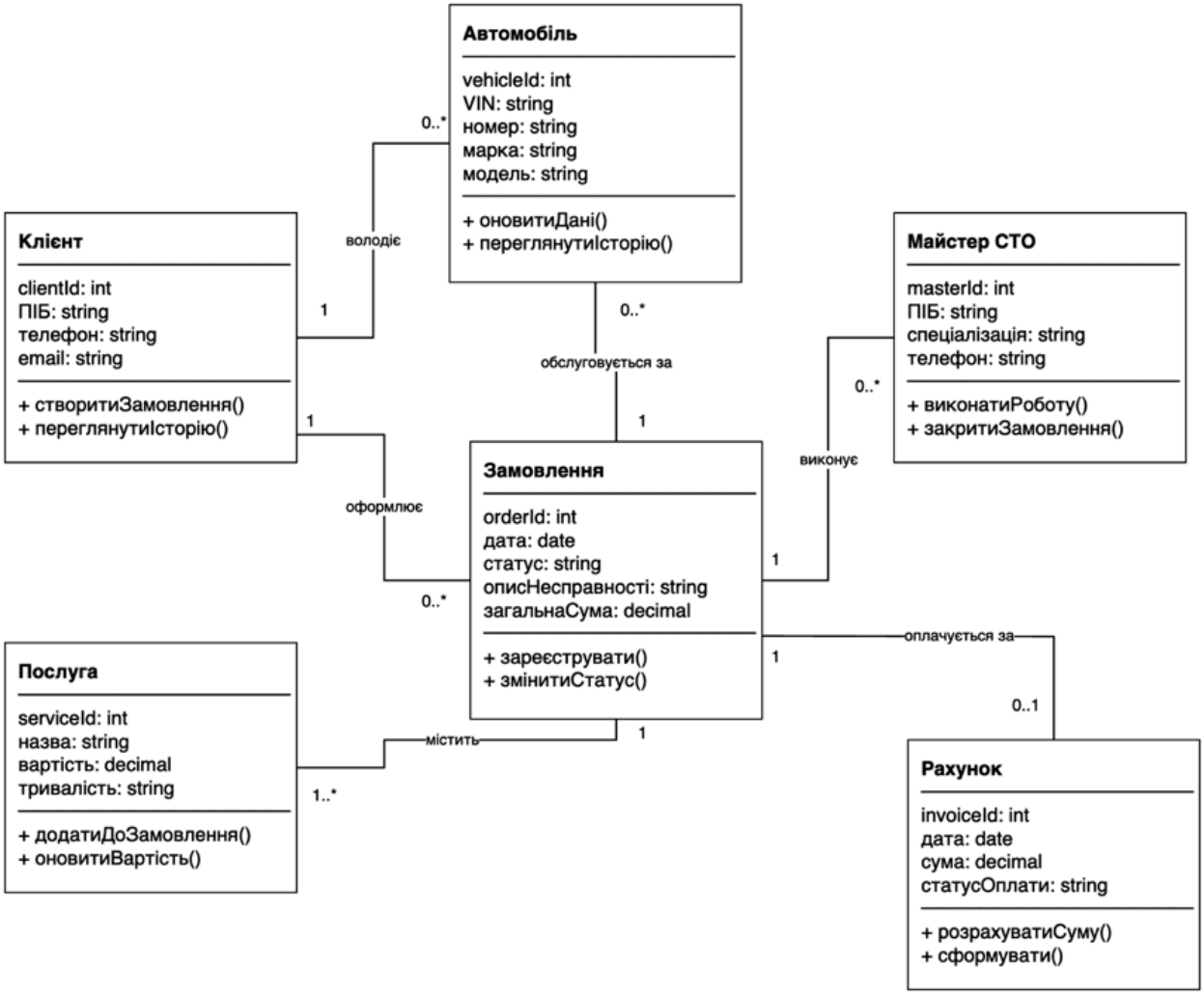

Рис. 2.2. Діаграма класів облікової системи СТО

Центральним класом у моделі є Замовлення, оскільки саме він об’єднує інформацію про клієнта, автомобіль, майстра, перелік послуг, статус виконання та загальну суму. Клас Клієнт зберігає персональні й контактні дані власника автомобіля та забезпечує створення замовлення і перегляд історії обслуговування. Клас Автомобільмістить VIN, номерний знак, марку та модель транспортного засобу, а також методи оновлення даних і перегляду історії ремонтів.

Клас Майстер СТО відповідає за виконання робіт і зміну стану замовлення після завершення ремонту. Клас Послугаописує окрему сервісну операцію, її назву, вартість і тривалість. Клас Рахунок використовується для

фінансового оформлення виконаного замовлення, розрахунку суми та фіксації статусу оплати.

Основні класи програмної моделі наведено в табл. 2.2.

Таблиця 2.2 - Основні класи облікової системи СТО

Клас	Призначення
Клієнт	Зберігання даних клієнта, створення замовлення, перегляд історії
Автомобіль	Облік транспортного засобу, VIN, номера, марки та моделі
Замовлення	Реєстрація звернення, фіксація несправності, статусу та суми
Майстер СТО	Виконання ремонтних робіт і закриття замовлення
Послуга	Опис сервісної операції, її вартості та тривалості
Рахунок	Формування фінансового документа та контроль оплати

Між класами встановлено зв’язки з урахуванням логіки роботи СТО. Один клієнт може володіти кількома автомобілями, тому між класами Клієнт і Автомобіль задано зв’язок 1:0..*. Один клієнт може оформити декілька замовлень, а кожне замовлення належить одному клієнту. Автомобіль також може мати багато замовлень протягом періоду експлуатації, що дозволяє формувати історію обслуговування.

Клас Замовлення пов’язаний із класом Послуга, оскільки в одному замовленні може бути одна або декілька сервісних операцій. Зв’язок із класом Майстер СТО показує, що майстер виконує замовлення та змінює його статус. Зв’язок між Замовленням і Рахунком має кратність 1:0..1, оскільки рахунок формується після погодження або завершення замовлення.

Для уточнення взаємодії між класами було розроблено три діаграми кооперації. Перша кооперація, подана на рис. 2.3, описує процес створення замовлення на обслуговування. Клієнт передає дані про себе та автомобіль,

автомобіль прив’язується до нового замовлення, після чого майстер приймає замовлення до роботи.

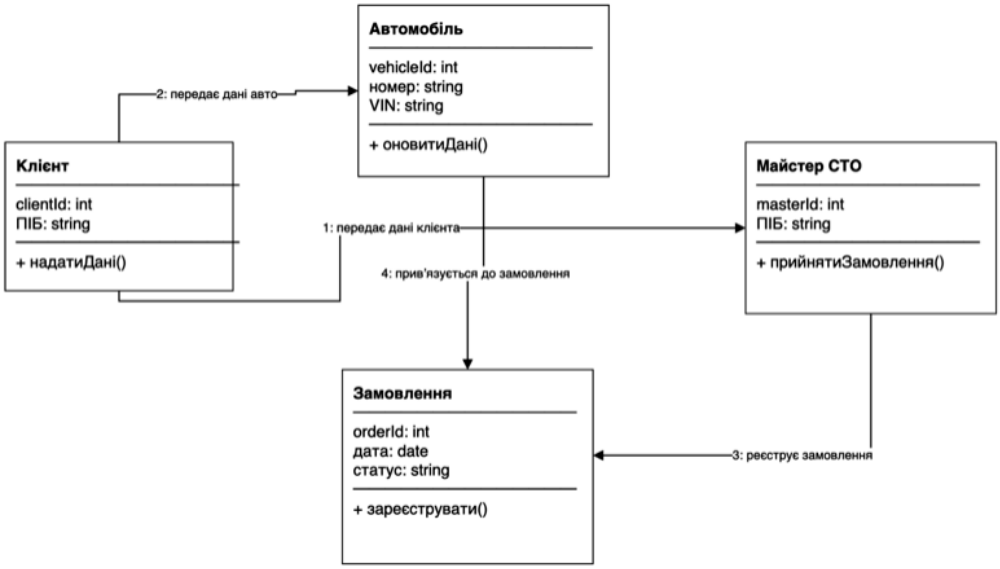

Рис. 2.3. Кооперація створення замовлення на обслуговування

Друга кооперація відображає процес виконання ремонтної або сервісної роботи, рис. 2.4. Майстер приймає замовлення в роботу, додає необхідну послугу, після чого послуга фіксується у замовленні. Після виконання робіт змінюється статус замовлення та оновлюється історія обслуговування автомобіля.

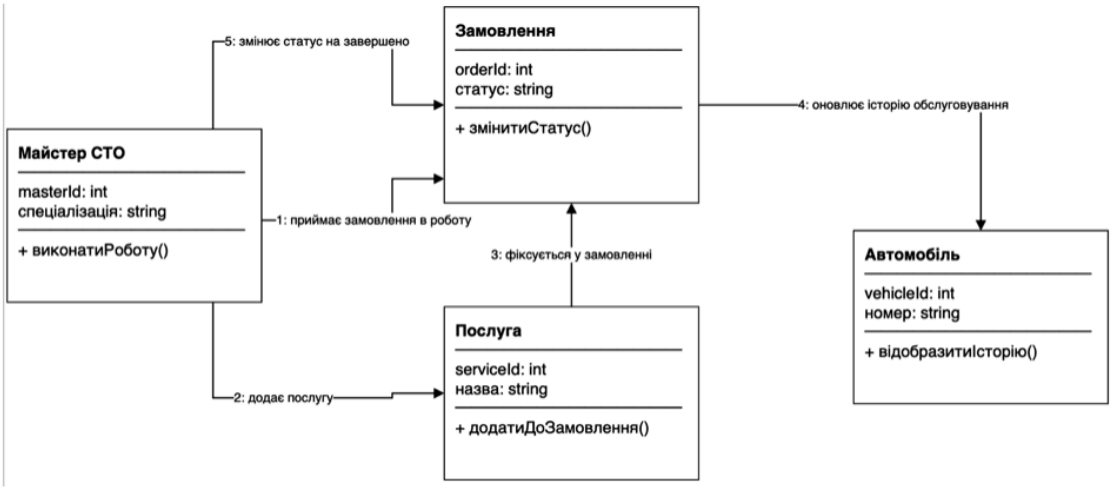

Рис. 2.4. Кооперація виконання сервісної роботи

Третя кооперація, наведена на рис. 2.5, описує формування рахунку за виконане замовлення. Замовлення передає дані про виконані послуги,

клас Послуга повертає їх вартість, після чого клас Рахунок формує фінансовий документ і передає його клієнту для оплати.

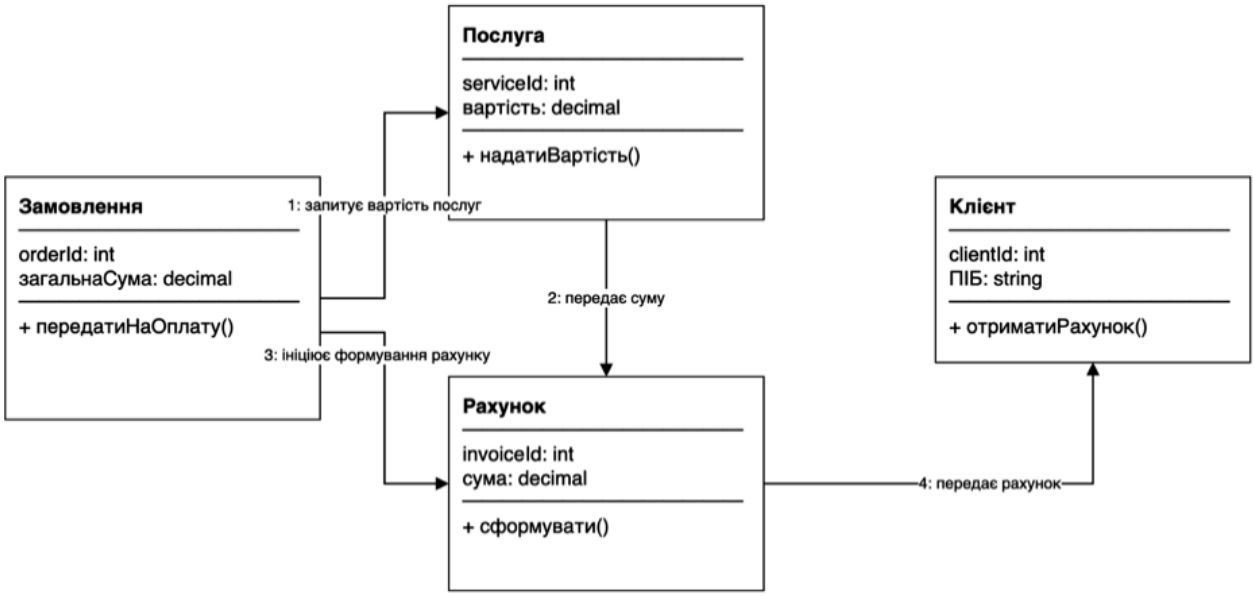

Рис.2.5. Кооперація формування рахунку за замовленням

Розроблена діаграма класів і кооперації показують, що програмна модель системи побудована навколо замовлення-наряду як основного об’єкта обліку. Такий підхід забезпечує логічний зв’язок між клієнтом, автомобілем, майстром, послугами та рахунком, а також створює основу для подальшої реалізації модулів реєстрації звернень, обліку ремонтів, фінансового оформлення і збереження історії обслуговування автомобілів.

2.3 Діаграма компонентів розроблювальної системи

Діаграма компонентів облікової системи СТО подана на рис. 2.6. Вона відображає склад програмних модулів, їх призначення та взаємодію з базою даних. Компонентна модель потрібна для розділення системи на окремі функціональні блоки, які відповідають за облік клієнтів, автомобілів, замовлень, послуг, запчастин, рахунків та оплат.

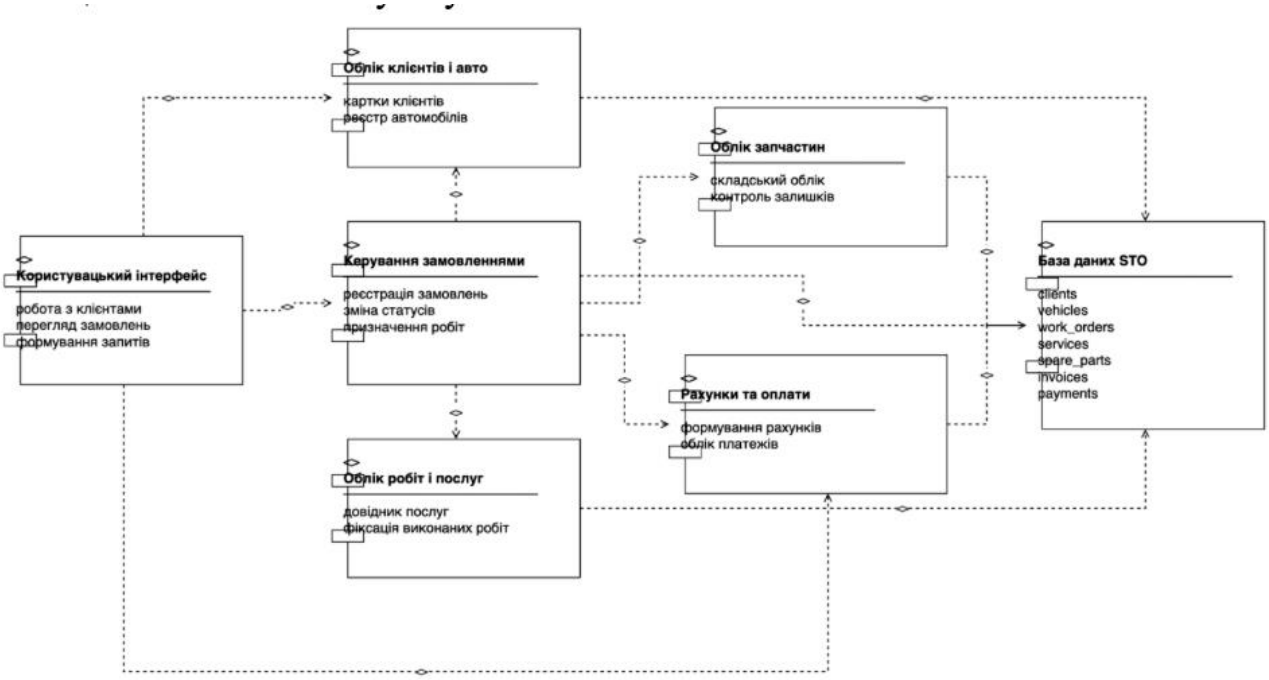

Рис. 2.6. Діаграма компонентів облікової системи СТО

Основним вхідним компонентом є Користувацький інтерфейс. Через нього працівник СТО реєструє клієнтів, створює замовлення, переглядає автомобілі, додає виконані роботи, формує рахунки та отримує звітну інформацію. Інтерфейс взаємодіє з прикладними модулями, а не звертається до бази даних напряму.

Центральним компонентом є Керування замовленнями, оскільки він об’єднує більшість бізнес-процесів СТО. Через нього створюється замовлення-наряд, змінюється статус ремонту, призначаються роботи та фіксується виконання сервісних операцій. Для роботи цього компонента використовуються дані з модулів клієнтів, автомобілів, послуг, запчастин і фінансового обліку.

Компоненти Облік клієнтів і авто, Облік робіт і послуг, Облік запчастин, Рахунки та оплати реалізують окремі частини прикладної логіки. Вони забезпечують ведення довідників, контроль залишків, фіксацію виконаних робіт, списання деталей, формування рахунків і реєстрацію платежів. Уся інформація зберігається в компоненті База даних СТО, який містить

таблиці clients, vehicles, work_orders, services, spare_parts, invoices, payments.

Таблиця 2.3 - Призначення компонентів облікової системи СТО

Компонент	Призначення
Користувацький інтерфейс	Робота з клієнтами, замовленнями, рахунками та звітами
Облік клієнтів і авто	Зберігання карток клієнтів і реєстру автомобілів
Керування замовленнями	Створення замовлень-нарядів, зміна статусів, призначення робіт
Облік робіт і послуг	Ведення переліку послуг, вартості та фіксація виконаних робіт
Облік запчастин	Складський облік, контроль залишків, списання деталей у замовлення
Рахунки та оплати	Формування рахунків, облік платежів, контроль статусу оплати
База даних СТО	Централізоване зберігання основних даних системи

Запропонована компонентна структура забезпечує логічне розділення функцій системи. Кожен модуль виконує окрему задачу, але всі вони пов’язані через замовлення-наряд і спільну базу даних. Такий підхід спрощує подальшу реалізацію, тестування та розширення системи, зокрема додавання аналітики, повідомлень клієнтам або інтеграції з бухгалтерськими сервісами.

2.4 Діаграма пакетів розроблювальної системи

Діаграма пакетів облікової системи СТО подана на рис. 2.7. Вона відображає логічне групування програмних модулів за функціональним призначенням і показує залежності між інтерфейсом, прикладною логікою, доменними моделями, аналітикою, автентифікацією та інфраструктурою даних.

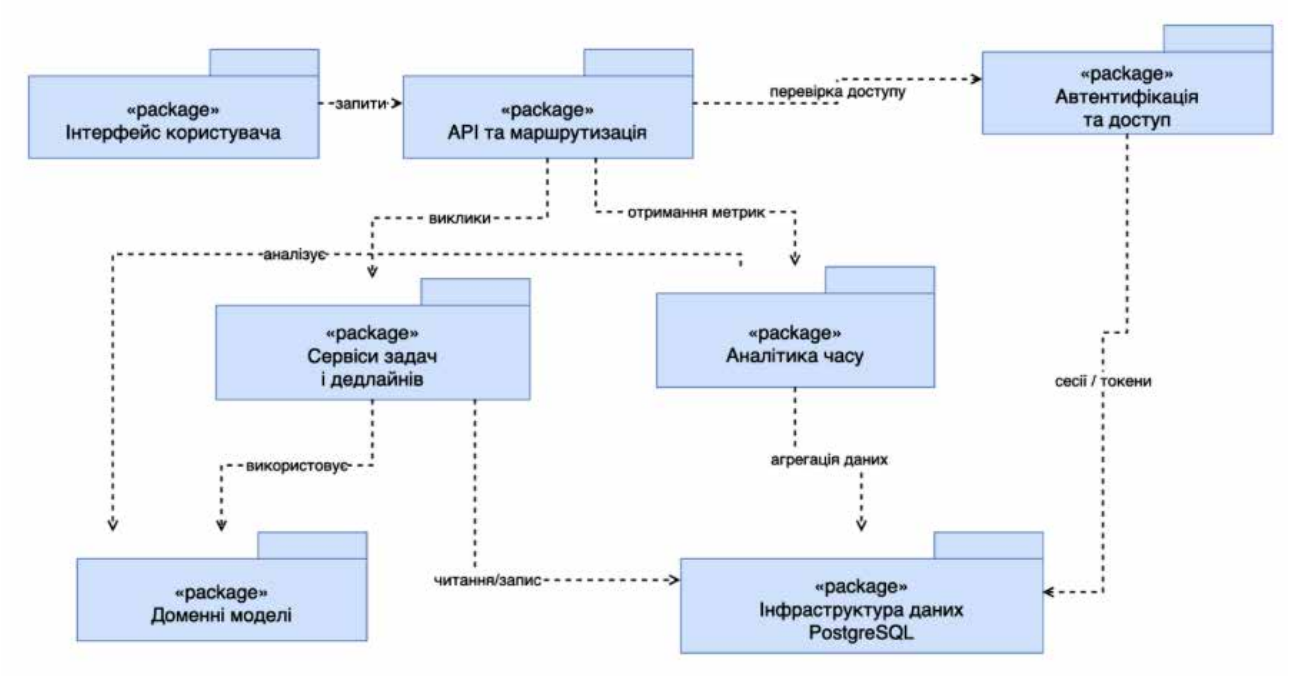

Рис. 2.7. Діаграма пакетів облікової системи СТО

Пакет Інтерфейс користувача містить екранні форми для роботи менеджера, майстра, бухгалтера та адміністратора. Через нього виконуються операції з клієнтами, автомобілями, замовленнями-нарядами, послугами, запчастинами, рахунками та звітами. Цей пакет не працює з базою даних напряму, а передає запити до пакета API та маршрутизація.

Пакет API та маршрутизація відповідає за приймання запитів від інтерфейсу, перевірку параметрів, вибір потрібного сервісу та повернення результату користувачу. Через нього здійснюється доступ до функцій реєстрації клієнтів, створення замовлень, оновлення статусів, формування рахунків і перегляду аналітики.

Пакет Автентифікація та доступ забезпечує перевірку користувача, створення сесії або токена та контроль ролей. Це дає змогу обмежити доступ до фінансових операцій, керування працівниками, редагування замовлень і перегляду адміністративних звітів.

Пакет Сервіси СТО містить основну бізнес-логіку системи: оброблення замовлень-нарядів, призначення майстрів, додавання послуг, списання запчастин, зміну статусів ремонту та підготовку даних для рахунків. Він

використовує пакет Доменні моделі, де описані основні об’єкти предметної області.

Пакет Аналітика СТО призначений для розрахунку показників роботи сервісу: кількості замовлень, завантаженості майстрів, вартості виконаних робіт, використання запчастин, суми оплат і статусів замовлень. Агреговані дані отримуються з пакета інфраструктури даних.

Пакет Інфраструктура даних відповідає за роботу з базою даних, репозиторіями та запитами до таблиць clients, vehicles, work_orders, services, spare_parts, invoices, payments. Саме цей пакет забезпечує збереження, читання й оновлення інформації.

Таблиця 2.4 - Призначення пакетів програмної системи СТО

Пакет	Призначення
Інтерфейс користувача	Форми для роботи з клієнтами, авто, замовленнями, рахунками та звітами
API та маршрутизація	Оброблення запитів і передавання їх до відповідних сервісів
Автентифікація та доступ	Вхід у систему, сесії, токени, перевірка ролей користувачів
Сервіси СТО	Бізнес-логіка замовлень, послуг, запчастин, статусів і рахунків
Аналітика СТО	Формування показників завантаження, оплат, робіт і використання запчастин
Доменні моделі	Опис основних сутностей: клієнт, автомобіль, замовлення, послуга, рахунок
Інфраструктура даних	Репозиторії, запити до бази даних, збереження та оновлення даних

Запропонована пакетна структура розділяє систему на незалежні логічні частини. Це спрощує розробку, тестування й подальше розширення програмного забезпечення. Наприклад, аналітичний модуль або модуль сповіщень можна доопрацювати без зміни основної логіки обліку замовлень, а зміни в базі даних можна ізолювати в пакеті інфраструктури даних.

2.5 Висновки до другого розділу

У другому розділі виконано проєктування інформаційного та програмного забезпечення облікової системи для станції технічного обслуговування. На основі результатів аналізу предметної області було визначено структуру даних, основні програмні класи, компоненти та пакети системи.

Розроблено логічну модель даних у вигляді ER-діаграми, яка відображає зв’язки між клієнтами, автомобілями, працівниками, замовленнями-нарядами, послугами, запчастинами, рахунками та оплатами. Модель побудовано з урахуванням принципів нормалізації, що дозволяє зменшити дублювання інформації, забезпечити цілісність даних і сформувати основу для подальшої реалізації фізичної структури бази даних.

Побудовано UML-діаграму класів, у якій центральним об’єктом визначено клас `Замовлення`. Саме він поєднує дані про клієнта, автомобіль, майстра, виконані послуги та рахунок. Розроблені кооперації уточнюють логіку взаємодії класів під час створення замовлення, виконання сервісної роботи та формування рахунку. Це дає змогу чітко визначити обов’язки основних об’єктів програмної системи.

Також розроблено діаграму компонентів, яка показує поділ системи на функціональні модулі: користувацький інтерфейс, облік клієнтів і автомобілів, керування замовленнями, облік робіт і послуг, складський облік запчастин, рахунки та оплати, базу даних СТО. Така структура забезпечує модульність системи та спрощує її реалізацію, тестування й подальше розширення.

Діаграма пакетів відображає логічну організацію програмного забезпечення за рівнями: інтерфейс користувача, API та маршрутизація, автентифікація і доступ, сервіси СТО, доменні моделі, аналітика та інфраструктура даних. Це дозволяє відокремити інтерфейсну частину від

бізнес-логіки та роботи з базою даних, що підвищує керованість архітектури системи.

Отже, у другому розділі сформовано проєктну основу для реалізації облікової системи СТО. Розроблені моделі визначають структуру даних, програмні об’єкти, функціональні компоненти та логіку взаємодії між модулями. Отримані результати є базою для подальшого вибору технологічного стеку, створення фізичної моделі бази даних і реалізації програмних модулів системи.

3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ ТА РОЗРОБЛЕННЯ АЛГОРИТМІЧНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Обґрунтування вибору технологічних засобів та інструментів реалізації

Для реалізації комплексного програмного забезпечення облікової системи для станції технічного обслуговування було обрано технологічний стек, орієнтований на просте розгортання, стабільну роботу з локальною базою даних, зручний графічний інтерфейс та можливість швидкого доопрацювання функціональних модулів. Оскільки система призначена для автоматизації внутрішніх процесів СТО, доцільним є використання настільного застосунку з локальним зберіганням даних.

Як основну мову програмування обрано Python. Вибір зумовлений простотою синтаксису, наявністю вбудованих засобів роботи з SQLite, підтримкою об’єктно-орієнтованого програмування та великою кількістю бібліотек для створення прикладних програм. Python дозволяє реалізувати бізнес-логіку системи без надмірного ускладнення структури проєкту.

Для побудови графічного інтерфейсу використано бібліотеку PyQt6. Вона забезпечує створення повноцінного настільного застосунку з формами введення, таблицями, вкладками, кнопками, діалоговими вікнами та графічними елементами. За допомогою PyQt6 реалізовано вікно авторизації, головний інтерфейс, модулі клієнтів, автомобілів, замовлень-нарядів, складу, рахунків, оплат, план-графіка, аналітики та адміністрування.

Для зберігання даних обрано SQLite. Це вбудована реляційна система керування базами даних, яка не потребує окремого серверного встановлення. Такий підхід є зручним для навчального програмного прототипу, оскільки база даних зберігається у вигляді одного локального файлу. SQLite забезпечує

роботу з таблицями, первинними та зовнішніми ключами, транзакціями й обмеженнями цілісності даних.

Основні технологічні засоби реалізації наведено в табл. 3.1.

Таблиця 3.1 - Технологічні засоби реалізації програмної системи

Засіб	Призначення в системі
Python	Реалізація прикладної логіки, оброблення даних, робота з базою
PyQt6	Побудова графічного інтерфейсу користувача
SQLite	Локальне зберігання клієнтів, авто, замовлень, послуг, запчастин, рахунків і оплат
sqlite3	Програмна взаємодія Python із базою SQLite
QSS	Оформлення інтерфейсу, кольорова палітра, стилі кнопок, таблиць і форм
hashlib, secrets	Хешування паролів і створення службових значень для авторизації
CSV	Експорт аналітичних показників у табличний формат
VS Code	Редагування коду, запуск і налагодження програми
venv	Ізоляція середовища виконання та залежностей проєкту

Архітектура реалізованого застосунку побудована за модульним принципом. У програмі виокремлено блок роботи з базою даних, блок авторизації, головне вікно системи та окремі інтерфейсні модулі. Такий підхід дозволяє розділити логіку зберігання даних, оброблення операцій і відображення результатів користувачу.

База даних містить таблиці для зберігання користувачів, клієнтів, автомобілів, працівників, послуг, запчастин, замовлень-нарядів, виконаних робіт, використаних деталей, рахунків, платежів, записів у план-графіку, контрольних перевірок якості, сповіщень та журналу подій. Для підтримки цілісності даних використано зовнішні ключі, які пов’язують замовлення з автомобілями, клієнтами, майстрами, послугами, запчастинами та рахунками.

Інтерфейс програми реалізовано у вигляді настільного застосунку з авторизацією та верхньою навігацією. Користувач після входу отримує доступ до основних розділів системи відповідно до своєї ролі. Для адміністратора передбачено модуль керування користувачами та перегляд журналу подій, для

менеджера: робота з клієнтами, автомобілями й замовленнями, для майстра: контроль виконання робіт, для бухгалтера: робота з рахунками та оплатами.

Використання PyQt6 дало змогу створити зручний графічний інтерфейс без потреби в браузері або вебсервері. Табличні компоненти використовуються для перегляду замовлень, клієнтів, автомобілів, складу, рахунків, платежів і аналітичних даних. Форми введення забезпечують додавання нових записів, зміну статусів, списання запчастин і формування рахунків.

Для оформлення інтерфейсу застосовано QSS-стилі. Це дозволило задати єдину палітру, вигляд кнопок, таблиць, карток показників, вкладок і форм. Окремо реалізовано інформаційні картки для відображення ключових показників: кількості замовлень, активних нарядів, критичних залишків на складі та суми отриманих оплат.

Для аналітичного модуля реалізовано прості графічні віджети, які відображають кількість замовлень за статусами, фінансові надходження та завантаження майстрів. Також передбачено експорт KPI-показників у CSV-файл, що дозволяє використовувати результати роботи системи для звітності або подальшого аналізу.

Обраний стек технологій відповідає поставленим вимогам до системи. Він забезпечує локальну роботу програми, простий запуск, надійне зберігання даних, зручну взаємодію користувача з функціями СТО та можливість подальшого розширення. У разі розвитку проєкту систему можна доповнити мережевим режимом роботи, резервним копіюванням бази даних, інтеграцією з SMS-сервісами, фіскальним модулем або вебверсією для клієнтів.

3.2 Практична реалізація бази даних системи

Практична реалізація бази даних облікової системи СТО виконана з використанням SQLite. Такий підхід відповідає обраному стеку реалізації, оскільки база даних зберігається у локальному файлі sto_manager.sqlite і не

потребує окремого серверного налаштування. Структура бази даних сформована відповідно до логічної ER-моделі, розробленої у другому розділі. Фізична модель бази даних подана на рис. 3.1.

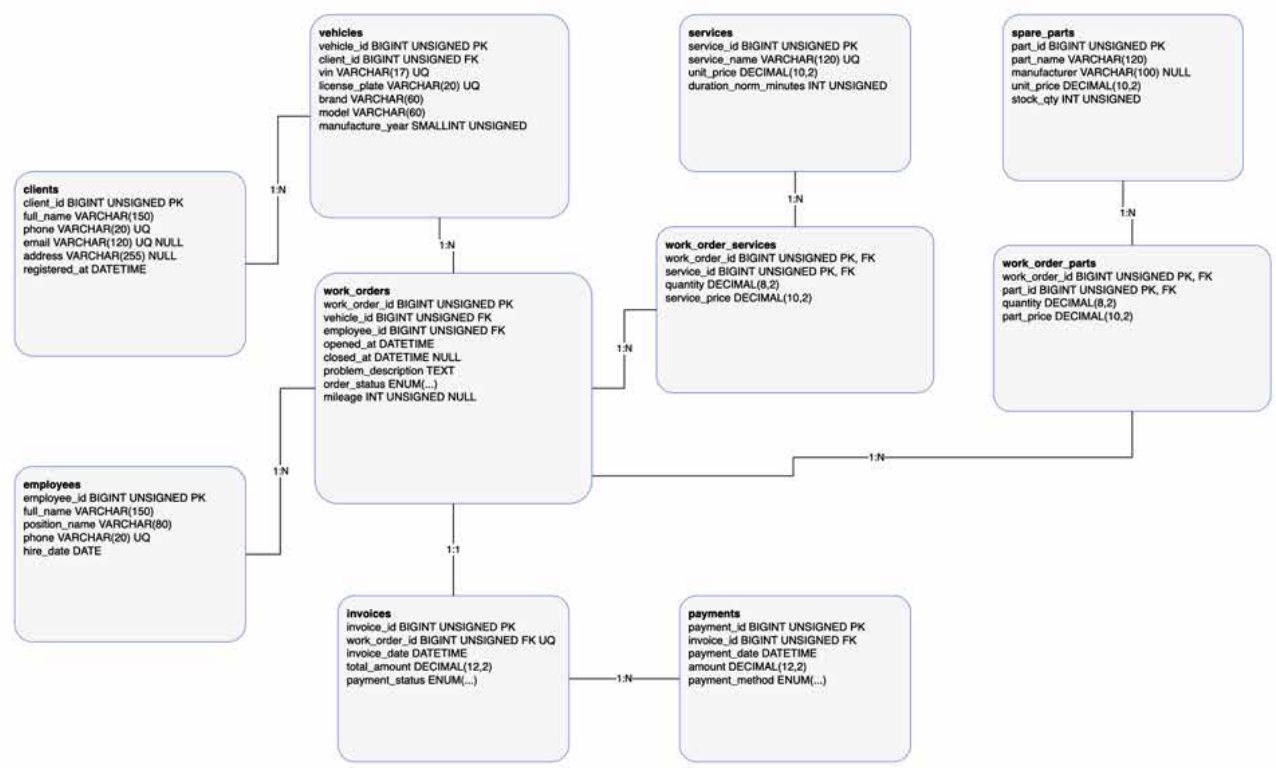

Рис.3.1. Фізична модель бази даних облікової системи СТО

Перед створенням таблиць у SQLite активується контроль зовнішніх ключів. Це необхідно для підтримки цілісності зв'язків між клієнтами, автомобілями, замовленнями, рахунками та оплатами.

```
PRAGMA foreign_keys = ON;
```

Основною таблицею для зберігання клієнтів є clients. Вона містить ПІБ, телефон, електронну пошту, адресу та дату реєстрації клієнта в системі.

```
CREATE TABLE clients (
    client_id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name TEXT NOT NULL,
    phone TEXT,
    email TEXT UNIQUE,
    address TEXT,
    registered_at TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Дані про транспортні засоби зберігаються в таблиці vehicles. Кожен автомобіль прив'язується до конкретного клієнта через зовнішній ключ client_id. Це дозволяє одному клієнту мати декілька автомобілів і зберігати історію звернень по кожному з них.

```
CREATE TABLE vehicles (
    vehicle_id INTEGER PRIMARY KEY AUTOINCREMENT,
    client_id INTEGER NOT NULL,
    vin TEXT UNIQUE,
    license_plate TEXT NOT NULL UNIQUE,
    brand TEXT NOT NULL,
    model TEXT NOT NULL,
    manufacture_year INTEGER,
    mileage INTEGER DEFAULT 0,
    FOREIGN KEY (client_id) REFERENCES clients(client_id)
        ON DELETE CASCADE
);
```

Працівники СТО зберігаються в таблиці employees. Ця таблиця використовується для призначення відповідального майстра або менеджера на замовлення-наряд.

```
CREATE TABLE employees (
    employee_id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name TEXT NOT NULL,
    position_name TEXT NOT NULL,
    phone TEXT,
    hire_date TEXT
);
```

Центральною таблицею бази даних є work_orders, оскільки саме замовлення-наряд поєднує автомобіль, відповідального працівника, опис несправності, статус ремонту, пробіг і підсумкову вартість робіт.

```
CREATE TABLE work_orders (
    work_order_id INTEGER PRIMARY KEY AUTOINCREMENT,
    vehicle_id INTEGER NOT NULL,
    employee_id INTEGER,
    opened_at TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,
    closed_at TEXT,
    problem_description TEXT NOT NULL,
    order_status TEXT NOT NULL DEFAULT 'Прийнято',
    mileage INTEGER,
    total_amount REAL NOT NULL DEFAULT 0,
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id)
        ON DELETE CASCADE,
    FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
        ON DELETE SET NULL
);
```

Довідник сервісних робіт реалізовано через таблицю services. Вона містить назву послуги, нормативну тривалість та базову вартість. Винесення послуг в окрему таблицю усуває дублювання назв робіт у замовленнях.

```
CREATE TABLE services (
    service_id INTEGER PRIMARY KEY AUTOINCREMENT,
    service_name TEXT NOT NULL,
    unit_price REAL NOT NULL,
    duration_norm_minutes INTEGER NOT NULL
);
```

Запчастини зберігаються в таблиці spare_parts. Для кожної позиції фіксується назва, виробник, ціна та залишок на складі.

```
CREATE TABLE spare_parts (
    part_id INTEGER PRIMARY KEY AUTOINCREMENT,
    part_name TEXT NOT NULL,
    manufacturer TEXT,
    unit_price REAL NOT NULL,
    stock_qty INTEGER NOT NULL DEFAULT 0
);
```

Оскільки одне замовлення може містити декілька послуг, а одна послуга може використовуватись у багатьох замовленнях, зв'язок між work_orders і services реалізовано через проміжну таблицю work_order_services.

```
CREATE TABLE work_order_services (
    work_order_id INTEGER NOT NULL,
    service_id INTEGER NOT NULL,
    quantity REAL NOT NULL DEFAULT 1,
    service_price REAL NOT NULL,
    PRIMARY KEY (work_order_id, service_id),
    FOREIGN KEY (work_order_id) REFERENCES work_orders(work_order_id)
        ON DELETE CASCADE,
    FOREIGN KEY (service_id) REFERENCES services(service_id)
);
```

Аналогічно для списання запчастин у межах замовлення використовується таблиця work_order_parts. Вона зберігає кількість використаних деталей і фактичну ціну на момент додавання до замовлення.

```
CREATE TABLE work_order_parts (
    work_order_id INTEGER NOT NULL,
    part_id INTEGER NOT NULL,
    quantity INTEGER NOT NULL,
    part_price REAL NOT NULL,
    PRIMARY KEY (work_order_id, part_id),
    FOREIGN KEY (work_order_id) REFERENCES work_orders(work_order_id)
        ON DELETE CASCADE,
    FOREIGN KEY (part_id) REFERENCES spare_parts(part_id)
);
```

Фінансовий блок реалізовано таблицями invoices і payments. Таблиця invoices зберігає рахунок за замовленням, а payments дозволяє фіксувати одну або кілька оплат за одним рахунком.

```
CREATE TABLE invoices (
    invoice_id INTEGER PRIMARY KEY AUTOINCREMENT,
    work_order_id INTEGER NOT NULL UNIQUE,
    invoice_date TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,
    total_amount REAL NOT NULL,
    payment_status TEXT NOT NULL DEFAULT 'Не оплачено',
    FOREIGN KEY (work_order_id) REFERENCES work_orders(work_order_id)
        ON DELETE CASCADE
);
CREATE TABLE payments (
    payment_id INTEGER PRIMARY KEY AUTOINCREMENT,
    invoice_id INTEGER NOT NULL,
    payment_date TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,
    amount REAL NOT NULL,
    payment_method TEXT NOT NULL,
    FOREIGN KEY (invoice_id) REFERENCES invoices(invoice_id)
```

```
        ON DELETE CASCADE
);
```

Для початкового наповнення бази даних використовуються довідкові записи клієнтів, автомобілів, працівників, послуг і запчастин. Це дає змогу одразу перевірити роботу системи після першого запуску.

```
INSERT INTO clients (full_name, phone, email, address)
VALUES
('Бондаренко Максим', '+380501234567', 'bondarenko@gmail.com', 'м. Черкаси'),
('Мельник Олена', '+380671235555', 'melnyk@gmail.com', 'м. Черкаси');
INSERT INTO services (service_name, unit_price, duration_norm_minutes)
VALUES
('Діагностика автомобіля', 650.00, 60),
('Заміна моторної оливи', 500.00, 30),
('Заміна гальмівних колодок', 1200.00, 90);
INSERT INTO spare_parts (part_name, manufacturer, unit_price, stock_qty)
VALUES
('Моторна олива 5W-30', 'Shell', 420.00, 30),
('Фільтр масляний', 'Bosch', 280.00, 25),
('Гальмівні колодки передні', 'TRW', 1450.00, 9);
```

Під час створення замовлення-наряду система записує дані про автомобіль, відповідального працівника, опис проблеми, статус і пробіг. Початковий статус замовлення встановлюється як Прийнято.

```
INSERT INTO work_orders (
    vehicle_id,
    employee_id,
    problem_description,
    order_status,
    mileage
)
VALUES (
    1,
    1,
    'Планове ТО, заміна оливи та фільтрів',
    'Прийнято',
    168000
);
```

Додавання послуги до замовлення виконується через таблицю work_order_services. У ній зберігається не тільки ідентифікатор послуги, а й ціна на момент додавання, що дозволяє зберегти правильну історію вартості навіть після зміни довідника послуг.

```
INSERT INTO work_order_services (
    work_order_id,
    service_id,
    quantity,
    service_price
)
SELECT
    1,
    service_id,
    1,
    unit_price
```

```
FROM services
WHERE service_id = 2;
```

Після додавання запчастини до замовлення кількість на складі зменшується. Це забезпечує контроль залишків і дозволяє уникати списання деталей, яких фактично немає на складі.

```
INSERT INTO work_order_parts (
    work_order_id,
    part_id,
    quantity,
    part_price
)
SELECT
    1,
    part_id,
    1,
    unit_price
FROM spare_parts
WHERE part_id = 2;
UPDATE spare_parts
SET stock_qty = stock_qty - 1
WHERE part_id = 2
  AND stock_qty >= 1;
```

Підсумкова сума замовлення формується на основі вартості доданих послуг і використаних запчастин. Розрахована сума записується в таблицю work_orders.

```
UPDATE work_orders
SET total_amount =
    (
        SELECT COALESCE(SUM(quantity * service_price), 0)
        FROM work_order_services
        WHERE work_order_id = 1
    )
    +
    (
        SELECT COALESCE(SUM(quantity * part_price), 0)
        FROM work_order_parts
        WHERE work_order_id = 1
    )
WHERE work_order_id = 1;
```

Після завершення формування замовлення створюється рахунок. Його сума відповідає підсумковій вартості замовлення-наряду.

```
INSERT INTO invoices (
    work_order_id,
    total_amount,
    payment_status
)
SELECT
    work_order_id,
    total_amount,
    'Не оплачено'
FROM work_orders
WHERE work_order_id = 1;
```

Реалізована база даних відповідає вимогам нормалізації. Дані клієнтів, автомобілів, працівників, послуг, запчастин, замовлень, рахунків і оплат зберігаються в окремих таблицях. Проміжні таблиці work_order_services і work_order_parts усувають дублювання даних і забезпечують коректне представлення зв’язків типу багато-до-багатьох. Використання зовнішніх ключів забезпечує цілісність інформації, а збереження цін у таблицях замовлення дозволяє фіксувати фактичну вартість робіт і матеріалів на момент виконання ремонту.

3.3 Алгоритмізація програмних модулів системи

Алгоритмізація програмних модулів облікової системи СТО виконана для визначення послідовності дій, які забезпечують роботу основних функцій програмного забезпечення. У межах системи виділено алгоритми реєстрації клієнта та автомобіля, створення замовлення-наряду, додавання послуг і запчастин, формування рахунку та закриття замовлення після завершення робіт.

Алгоритм реєстрації клієнта та автомобіля використовується під час первинного звернення клієнта до СТО. Користувач вводить персональні дані клієнта, контактний номер, електронну пошту, а також інформацію про автомобіль: VIN, державний номер, марку, модель, рік випуску та пробіг. Після цього система перевіряє коректність введених даних і наявність дублювань за телефоном, VIN або номером автомобіля. Якщо дублювання виявлено, користувач отримує повідомлення про помилку. Якщо дані унікальні, система створює записи у таблицях clients і vehicles. Схему алгоритму наведено на рис. 3.2.

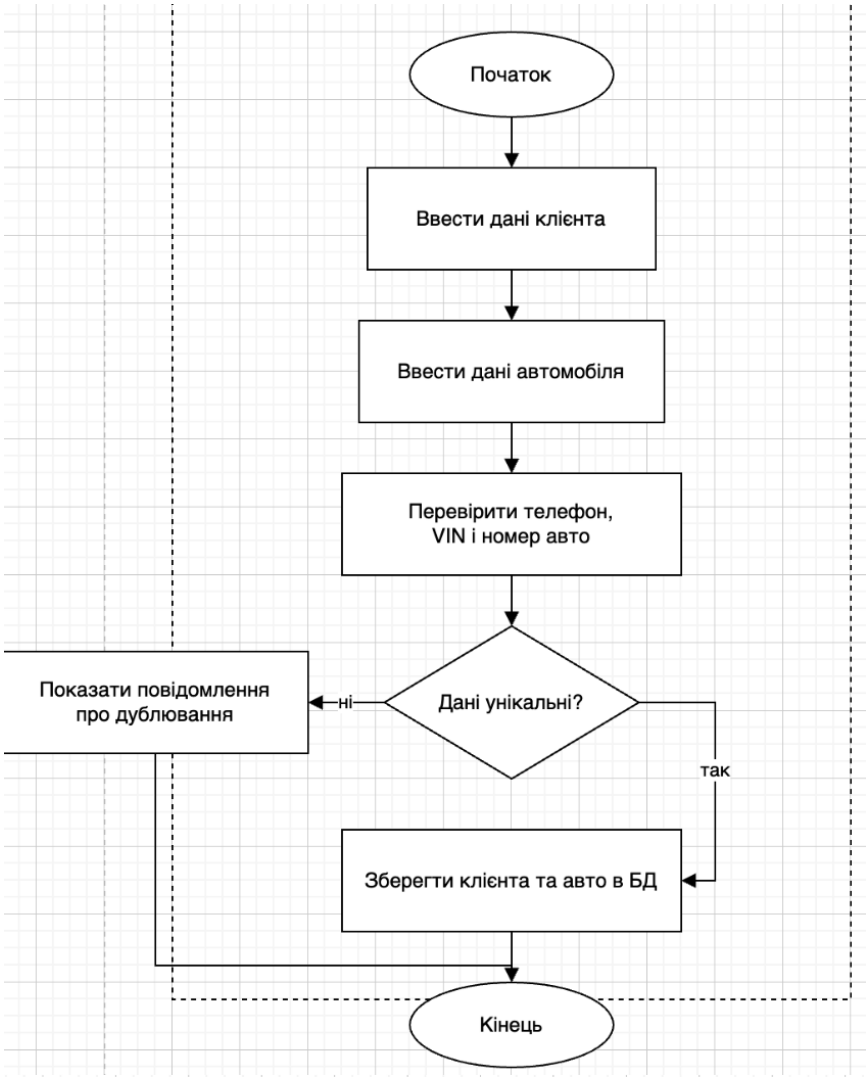

Рис. 3.2. Алгоритм реєстрації клієнта та автомобіля

Алгоритм створення замовлення-наряду починається з вибору клієнта та автомобіля. Далі користувач вказує опис несправності, поточний пробіг, пріоритет звернення та відповідального працівника. Система перевіряє, чи заповнені обов’язкові поля. Якщо дані неповні, користувач повертається до редагування форми. Якщо перевірка успішна, у таблиці work_orders створюється новий запис із початковим статусом Прийнято. Алгоритм створення замовлення-наряду подано на рис. 3.3.

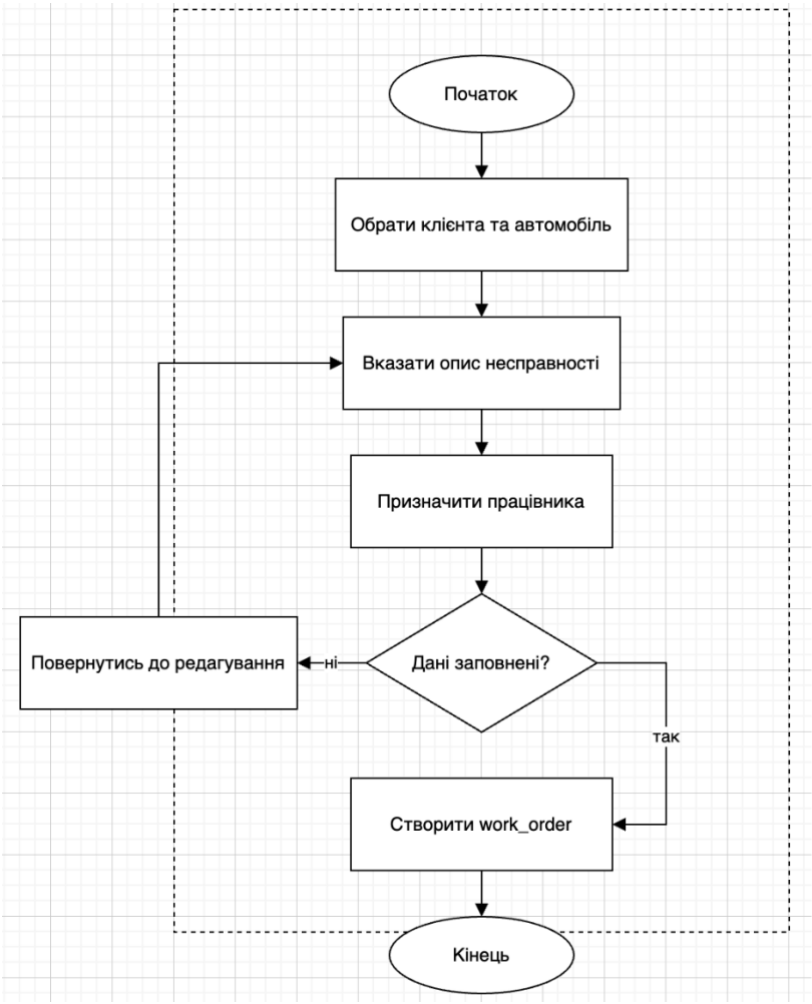

Рис. 3.3. Алгоритм створення замовлення-наряду

Алгоритм додавання послуг і запчастин забезпечує наповнення замовлення-наряду конкретними роботами та матеріалами. Користувач обирає замовлення, після чого додає сервісну послугу або складську позицію. Для послуг перевіряється наявність ціни та кількості. Для запчастин додатково перевіряється залишок на складі. Якщо дані некоректні або запчастини недостатньо, система показує повідомлення про помилку. Якщо перевірка пройдена, дані зберігаються у таблицях work_order_services або work_order_parts, а підсумкова сума замовлення автоматично перераховується. Алгоритм наведено на рис. 3.4.

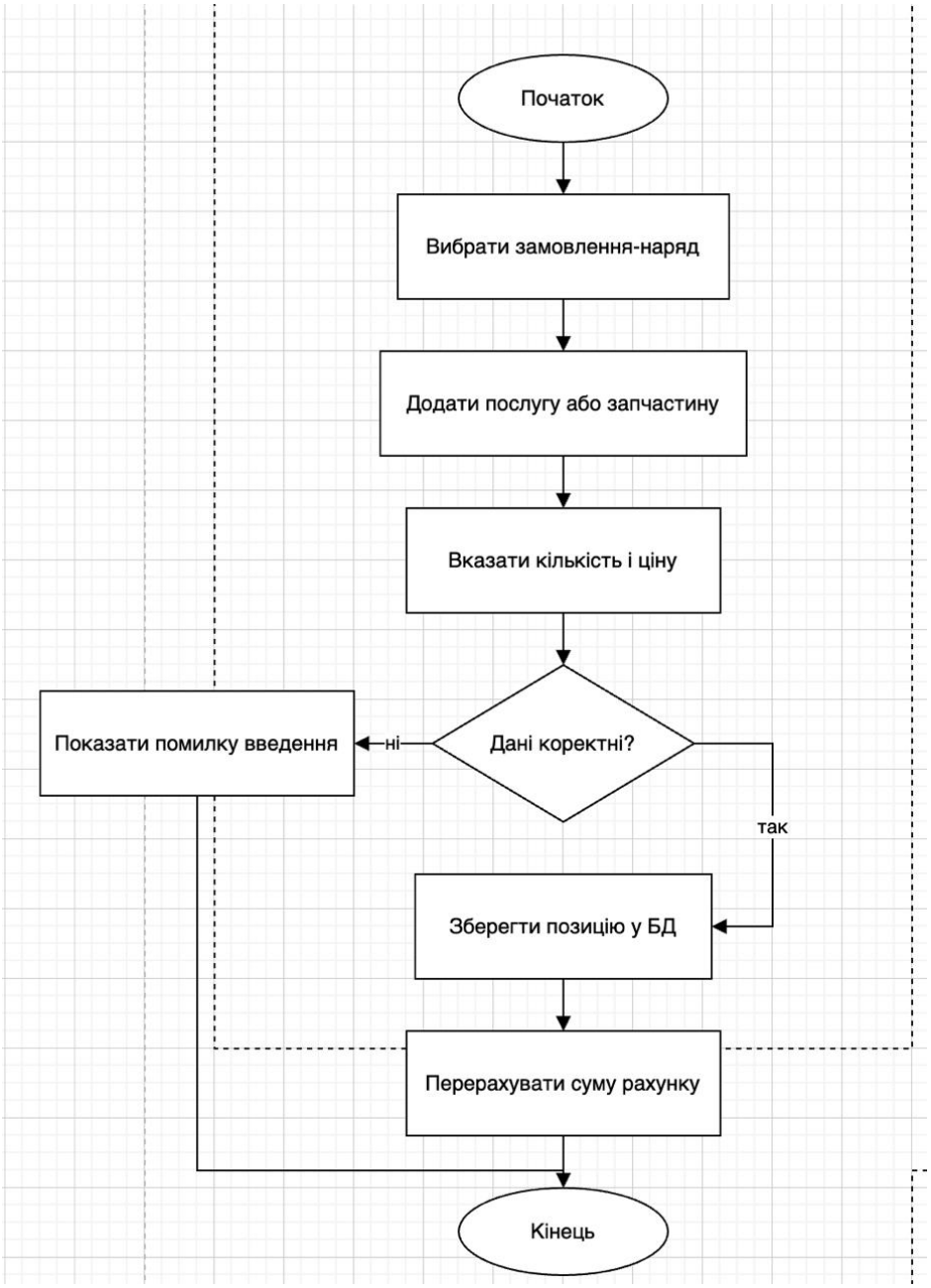

Рис. 3.4. Алгоритм додавання послуг і запчастин до замовлення

Алгоритм формування рахунку та обліку оплати використовується після внесення до замовлення виконаних робіт і використаних запчастин. Система отримує замовлення-наряд, розраховує загальну суму робіт і матеріалів, після чого створює рахунок у таблиці invoices. Якщо оплата не внесена, рахунок залишається зі статусом Не оплачено. Якщо платіж отримано, дані зберігаються у таблиці payments, а статус рахунку оновлюється відповідно до суми оплати. Якщо сума платежів покриває повну вартість рахунку, встановлюється статус Оплачено. Алгоритм подано на рис. 3.5.

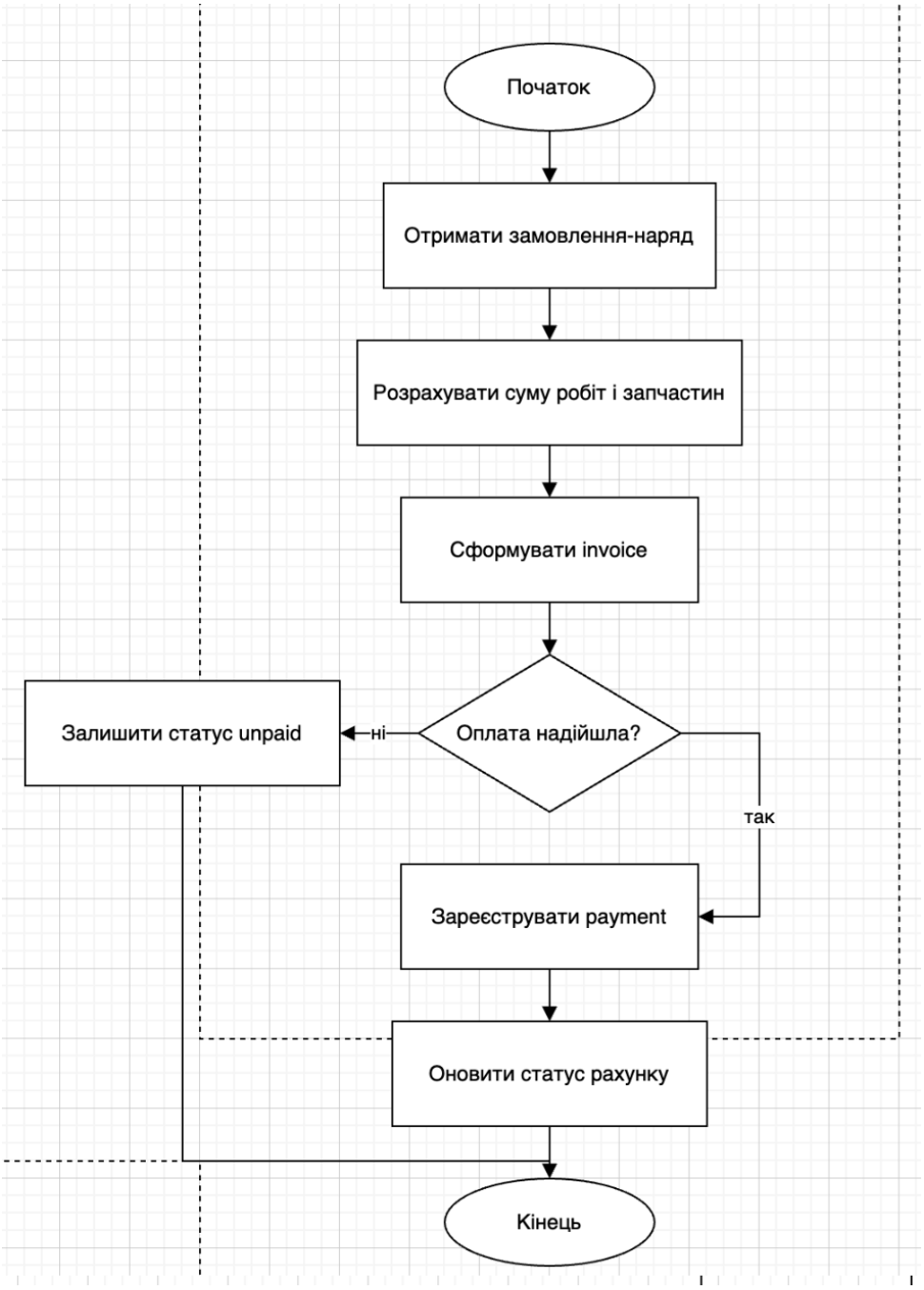

Рис. 3.5. Алгоритм формування рахунку та обліку оплати

Алгоритм закриття замовлення-наряду виконується після завершення ремонтних або сервісних робіт. Перед закриттям система перевіряє, чи виконано всі роботи та чи врегульовано фінансовий стан замовлення. Якщо є незавершені умови, користувач отримує повідомлення про неможливість закриття. Якщо всі умови виконані, у таблиці work_orders записується дата завершення closed_at, а статус замовлення змінюється на Завершено. Схему алгоритму наведено на рис. 3.6.

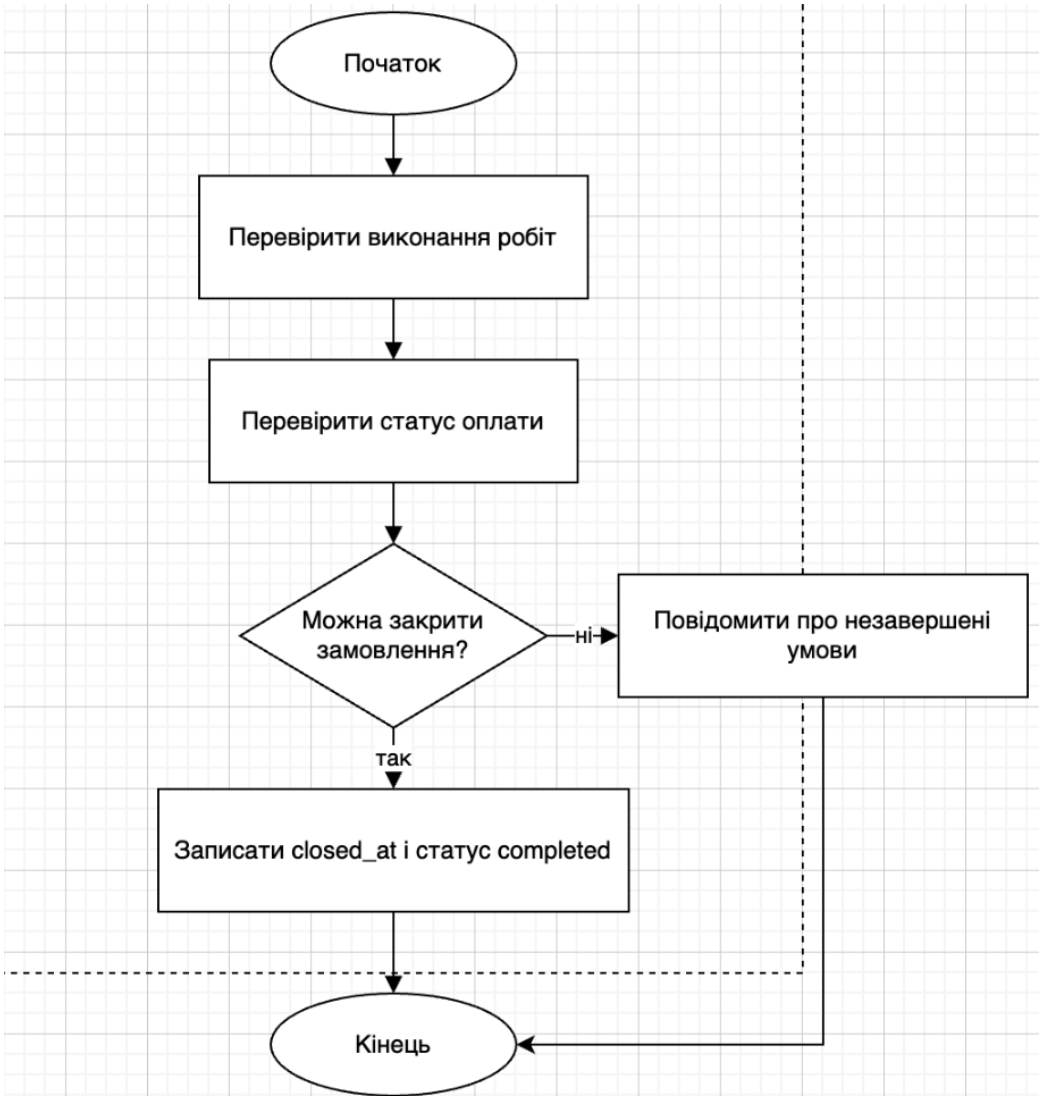

Рис. 3.6. Алгоритм закриття замовлення-наряду

Алгоритми побудовано з урахуванням перевірки вхідних даних, цілісності зв’язків між таблицями та необхідності уникнення помилкових операцій. Наприклад, замовлення не може бути створене без автомобіля, послуга або запчастина не додається без коректної кількості, а рахунок формується тільки на основі наявного замовлення-наряду. Це забезпечує надійність роботи системи та зменшує ризик некоректного збереження даних.

У програмній реалізації алгоритми пов’язані з відповідними модулями інтерфейсу. Модуль клієнтів і автомобілів відповідає за перший алгоритм, модуль замовлень реалізує створення та зміну статусів нарядів, модуль складу забезпечує роботу із запчастинами, а фінансовий модуль відповідає за рахунки

й оплати. Аналітична частина використовує уже сформовані дані для розрахунку показників роботи СТО.

Алгоритмізація програмних модулів забезпечує послідовну логіку функціонування системи: від реєстрації клієнта до завершення ремонту й фіксації оплати. Розроблені алгоритми стали основою для практичної реалізації основних функцій програмного забезпечення та подальшого тестування системи.

3.4 Висновки до третього розділу

У третьому розділі виконано проєктування архітектури програмної системи та розроблено алгоритмічне забезпечення облікової системи для станції технічного обслуговування. Основну увагу приділено вибору технологічних засобів, практичній реалізації бази даних і формалізації алгоритмів роботи основних програмних модулів.

Для реалізації програмного забезпечення обґрунтовано використання мови Python, бібліотеки PyQt6 та бази даних SQLite. Такий технологічний стек є доцільним для навчального програмного прототипу, оскільки забезпечує простий запуск, локальне зберігання даних, зручний графічний інтерфейс і достатню функціональність для автоматизації процесів СТО. Використання PyQt6 дозволило реалізувати настільний застосунок із формами введення, таблицями, вкладками, діалоговими вікнами, графічними елементами та навігацією між модулями.

Практична реалізація бази даних виконана відповідно до логічної моделі, розробленої в другому розділі. У базі SQLite створено таблиці для зберігання користувачів, клієнтів, автомобілів, працівників, послуг, запчастин, замовлень-нарядів, виконаних робіт, використаних матеріалів, рахунків, платежів, план-графіка, сповіщень, контрольних перевірок якості та журналу подій. Для

підтримки цілісності даних використано первинні та зовнішні ключі, а також зв’язки між основними таблицями системи.

У межах фізичної реалізації бази даних передбачено збереження фактичної вартості послуг і запчастин у момент додавання їх до замовлення-наряду. Це дозволяє зберігати історичну достовірність фінансових даних навіть у разі подальшої зміни цін у довідниках. Також реалізовано механізм перерахунку загальної суми замовлення після додавання робіт або списання запчастин зі складу.

Розроблено алгоритми роботи основних програмних модулів системи. До них належать алгоритми реєстрації клієнта та автомобіля, створення замовлення-наряду, додавання послуг і запчастин, формування рахунку, реєстрації оплати, закриття замовлення, формування аналітичних показників і журналювання дій користувачів. Запропоновані алгоритми забезпечують послідовну логіку роботи системи та враховують перевірку вхідних даних.

Особливу увагу приділено контролю коректності операцій. Замовлення не може бути створене без вибору автомобіля та опису проблеми, запчастина не списується за відсутності потрібного залишку на складі, рахунок формується тільки для існуючого замовлення, а оплата прив’язується до конкретного рахунку. Це знижує ризик помилкового збереження даних і забезпечує узгодженість інформації між модулями.

У результаті третього розділу сформовано практичну основу програмної реалізації системи. Обрано технологічні засоби, створено структуру бази даних, визначено основні SQL-операції та описано алгоритми роботи ключових модулів. Отримані результати є базою для подальшого тестування програмного забезпечення, оцінювання його працездатності та перевірки відповідності функціональним вимогам.

4 ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Тестування програмних модулів облікової системи для станції технічного обслуговування

Тестування програмних модулів комплексного програмного забезпечення облікової системи для станції технічного обслуговування виконувалося з метою перевірки коректності основних функцій, стабільності роботи застосунку, правильності збереження даних у базі SQLite та відповідності реалізованих модулів функціональним вимогам. Основна увага приділялася модулям авторизації, обліку клієнтів і автомобілів, створення замовлень-нарядів, додавання послуг і запчастин, формування рахунків, реєстрації оплат, планування записів, аналітики та адміністрування.

Тестування проводилося на рівні окремих програмних модулів та їх взаємодії між собою. Для перевірки використовувалися тестові облікові записи користувачів з різними ролями: адміністратор, менеджер, майстер і бухгалтер. Це дозволило перевірити не лише працездатність функцій, а й коректність розмежування доступу до окремих частин системи.

На першому етапі перевірявся модуль авторизації та реєстрації. Тестування включало вхід до системи з правильними та неправильними обліковими даними, створення нового користувача, перевірку унікальності email, збереження хешованого пароля та відкриття головного вікна після успішного входу. У результаті встановлено, що система коректно обробляє помилки введення, не допускає вхід з неправильним паролем і створює облікові записи з відповідною роллю.

Модуль обліку клієнтів і автомобілів перевірявся шляхом додавання нових клієнтів, внесення контактних даних, створення карток автомобілів, прив’язування транспортного засобу до конкретного клієнта та перегляду інформації у таблицях інтерфейсу. Окремо перевірено видалення клієнта разом

із пов’язаними автомобілями відповідно до встановлених зв’язків у базі даних. Тестування підтвердило, що зв’язок між таблицями clients і vehicles працює коректно.

Модуль замовлень-нарядів тестувався за сценарієм створення нового звернення клієнта. Користувач обирав автомобіль, відповідального працівника, пріоритет, пробіг і опис несправності. Система перевіряла заповнення обов’язкових полів і після успішної перевірки створювала запис у таблиці work_orders зі статусом Прийнято. Також перевірено зміну статусів замовлення на В роботі, Очікує запчастини, Завершено або Скасовано.

Модуль додавання послуг і запчастин перевірявся через наповнення замовлення-наряду сервісними роботами та складськими позиціями. Під час додавання послуги система фіксувала її кількість і вартість у таблиці work_order_services. Під час списання запчастини перевірявся залишок на складі. Якщо кількість була достатньою, позиція додавалася до work_order_parts, а залишок у spare_parts зменшувався. Якщо залишку було недостатньо, система виводила повідомлення про помилку.

Фінансовий модуль тестувався шляхом формування рахунку на основі замовлення-наряду. Система розраховувала загальну суму робіт і запчастин, створювала запис у таблиці invoices та встановлювала початковий статус Не оплачено. Після внесення платежу створювався запис у таблиці payments, а статус рахунку змінювався на Частково оплачено або Оплачено залежно від суми платежів.

План-графік записів перевірявся шляхом створення запланованих візитів клієнтів. Для кожного запису задавалися клієнт, автомобіль, майстер, дата, час, причина звернення та статус. У результаті тестування підтверджено, що записи коректно зберігаються в таблиці schedule і відображаються у відповідному модулі інтерфейсу.

Окремо перевірявся модуль контролю якості виконаних робіт. Для завершеного або активного замовлення створювався контрольний запис, у якому фіксувалися перевірка автомобіля, візуальний огляд, контроль

використаних запчастин, повідомлення клієнта та коментар. Дані зберігалися у таблиці quality_checks і могли бути використані для підтвердження завершення сервісного процесу.

Модуль аналітики перевірявся через формування KPI-показників: кількості замовлень, активних нарядів, завершених робіт, суми оплат, середнього чека, завантаження майстрів і критичних залишків на складі. Також перевірено експорт аналітичної інформації у CSV-файл. Отримані результати відповідали даним, що зберігаються у таблицях work_orders, invoices, payments, employees і spare_parts.

Загальний план тестування програмних модулів наведено у таблиці 4.1.

Таблиця 4.1 - План тестування програмних модулів облікової системи СТО

№	Програмний модуль	Тестові дії	Очікуваний результат
1	Авторизація та реєстрація	Вхід з правильними і неправильними даними, створення нового користувача	Коректний вхід до системи, відхилення неправильних даних, створення акаунта
2	Облік клієнтів	Додавання, перегляд і видалення клієнта	Дані клієнта зберігаються у таблиці clients
3	Облік автомобілів	Додавання авто до вибраного клієнта	Автомобіль зберігається у vehicles і прив’язується до клієнта
4	Замовлення-наряди	Створення нового замовлення з описом несправності	У таблиці work_orders створюється запис зі статусом Прийнято
5	Зміна статусу ремонту	Оновлення статусу замовлення	Статус змінюється, за потреби створюється сповіщення клієнту
6	Послуги у замовленні	Додавання сервісної роботи до наряду	Послуга зберігається у work_order_services, сума наряду оновлюється
7	Списання запчастин	Додавання запчастини до замовлення	Запчастина зберігається у work_order_parts, складський залишок зменшується
8	Складський облік	Додавання нової запчастини та контроль мінімального залишку	Дані зберігаються у spare_parts, критичні залишки відображаються в аналітиці
9	Рахунки	Формування рахунку за замовленням	Створюється запис у invoices з правильною сумою

Продовження таблиці 4.1

10	Оплати	Реєстрація часткової або повної оплати	Створюється запис у payments, статус рахунку оновлюється
11	План-графік	Додавання запису клієнта на дату і час	Запис зберігається у schedule
12	Контроль якості	Створення контрольного запису по замовленню	Дані зберігаються у quality_checks
13	Аналітика KPI	Перегляд дашборду, графіків і показників	Показники формуються на основі актуальних даних
14	Адміністрування	Зміна ролі або статусу користувача	Дані користувача оновлюються у users
15	Журнал подій	Виконання важливих операцій у системі	Події фіксуються у таблиці audit_log

Під час тестування також перевірялася робота обмежень цілісності бази даних. Замовлення не може бути створене без існуючого автомобіля, рахунок не створюється без замовлення-наряду, платіж не може бути прив’язаний до неіснуючого рахунку, а запчастина не списується, якщо її залишок на складі недостатній. Це підтверджує коректність реалізації зв’язків між таблицями та правильність використання зовнішніх ключів.

Результати тестування програмних модулів показали, що система виконує основні функції відповідно до сформованих вимог. Програмні модулі взаємодіють між собою послідовно: дані клієнтів і автомобілів використовуються при створенні замовлень, замовлення наповнюються послугами та запчастинами, після чого на їх основі формуються рахунки, оплати й аналітичні показники. Це забезпечує повний цикл обліку сервісних процесів СТО.

4.2 Тестування інтерфейсних модулів облікової системи для станції технічного обслуговування

Тестування інтерфейсних модулів облікової системи СТО виконувалося з метою перевірки зручності роботи користувача, коректності відображення

даних, правильності переходів між розділами та відповідності інтерфейсу функціональним можливостям системи. Перевірка проводилася для основних екранів програми: авторизації, реєстрації, дашборду, обліку клієнтів і автомобілів, замовлень-нарядів, складу, фінансів та KPI-аналітики.

На першому етапі перевірено вікно входу до системи, подане на рис. 4.1. Інтерфейс містить поля для введення email і пароля, вкладки входу та реєстрації, а також службову інформацію про тестові облікові записи. Під час тестування встановлено, що система коректно приймає правильні облікові дані, відкриває головне вікно відповідно до ролі користувача та не допускає вхід із неправильним паролем.

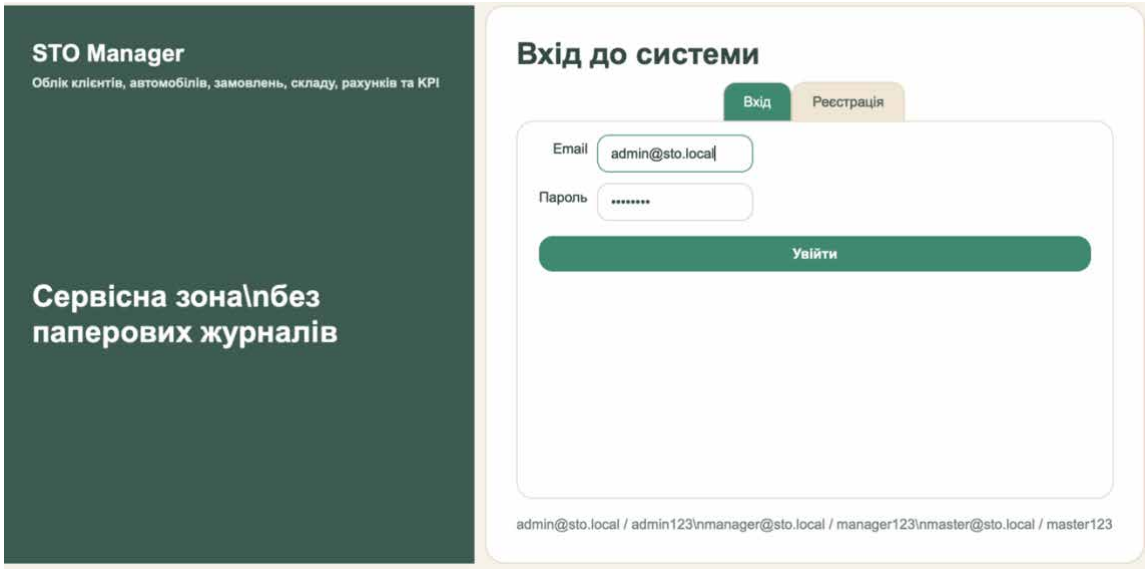

Рис. 4.1. Вікно входу до облікової системи СТО

Форма реєстрації користувача наведена на рис. 4.2. У ній передбачено введення ПІБ, email, пароля та вибір ролі користувача. Під час перевірки встановлено, що новий користувач створюється після заповнення обов’язкових полів, а система не допускає реєстрацію з некоректними або неповними даними.

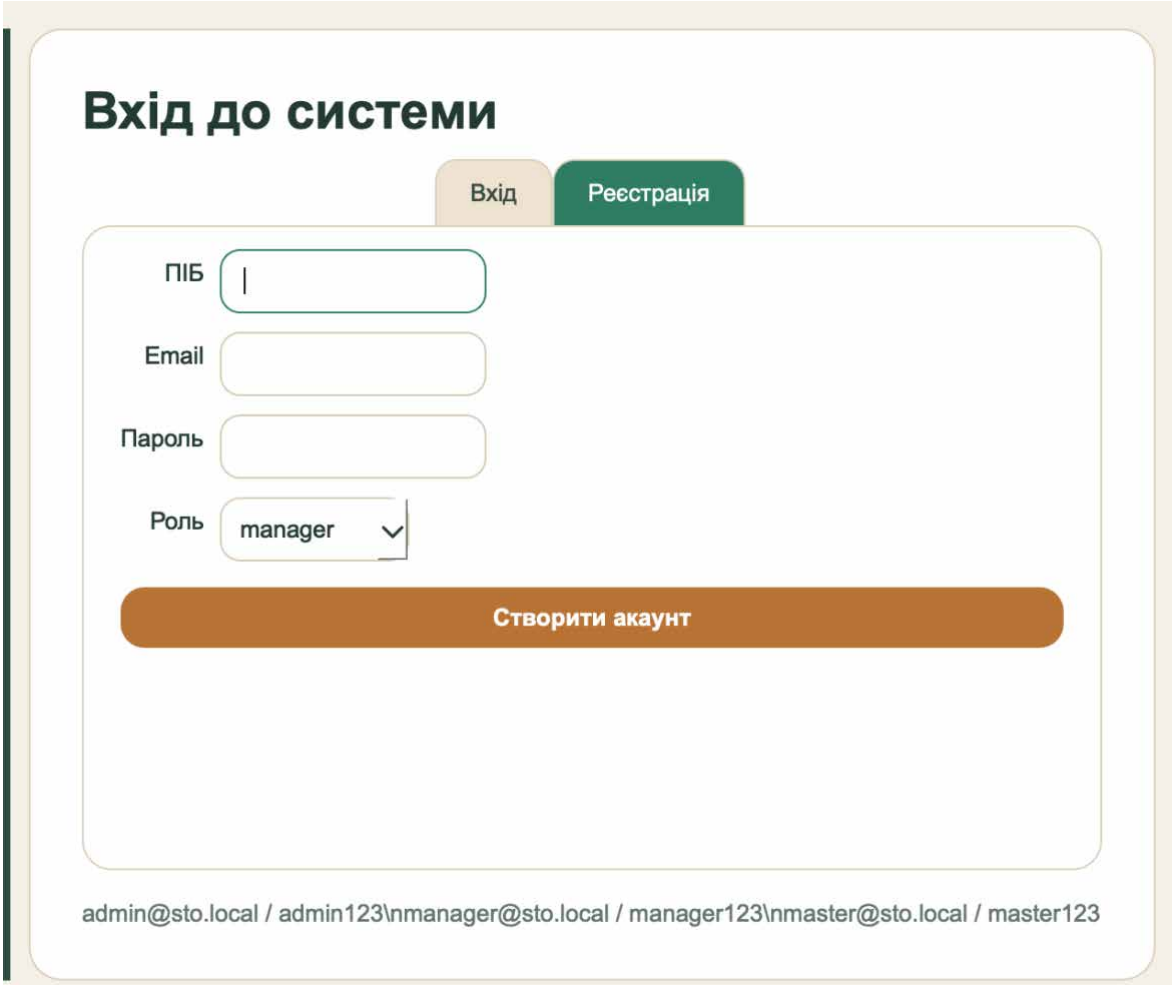

Рис. 4.2. Інтерфейс реєстрації користувача

Після успішної авторизації відкривається головний екран системи, поданий на рис. 4.3. Дашборд відображає загальну кількість нарядів, активні замовлення, критичні складські залишки та суму отриманих оплат. Також на екрані показано графіки за статусами замовлень і оплатами за днями. Під час тестування перевірено, що показники змінюються після створення нового замовлення, додавання оплати або списання запчастин.

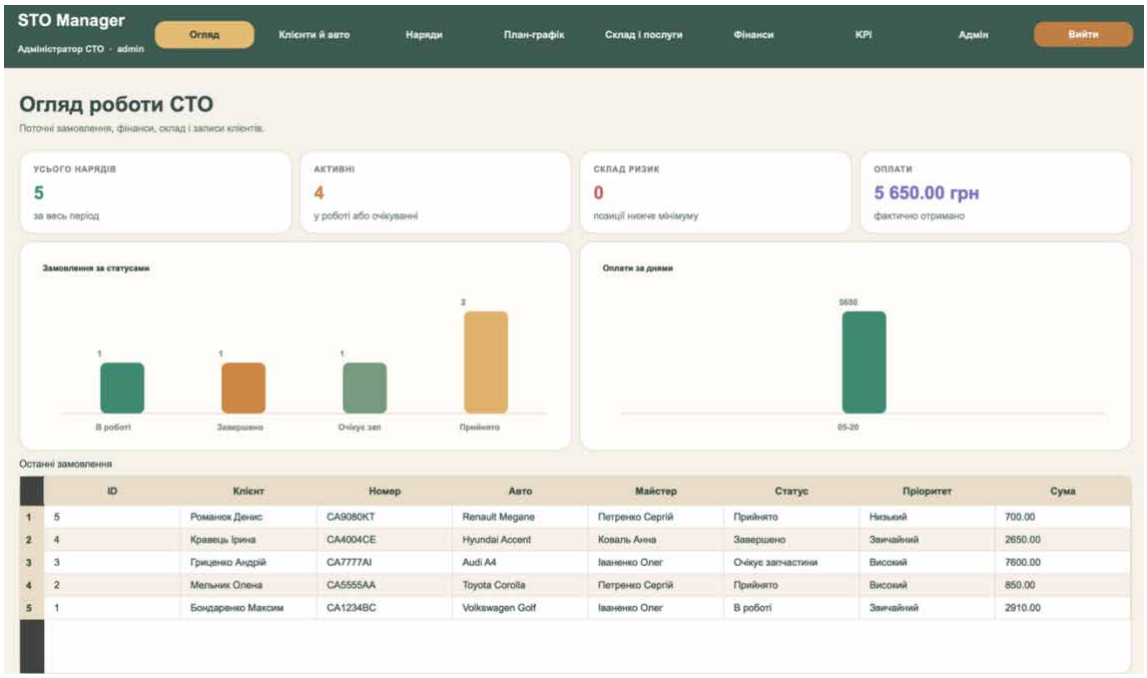

Рис. 4.3. Головний екран системи з показниками роботи СТО

Інтерфейс обліку клієнтів наведено на рис. 4.4. Форма дає змогу ввести ПІБ, телефон, email, адресу та примітки про клієнта. Після натискання кнопки додавання дані зберігаються в базі та відображаються у таблиці. Перевірка показала, що записи додаються коректно, а таблиця оновлюється без перезапуску програми.

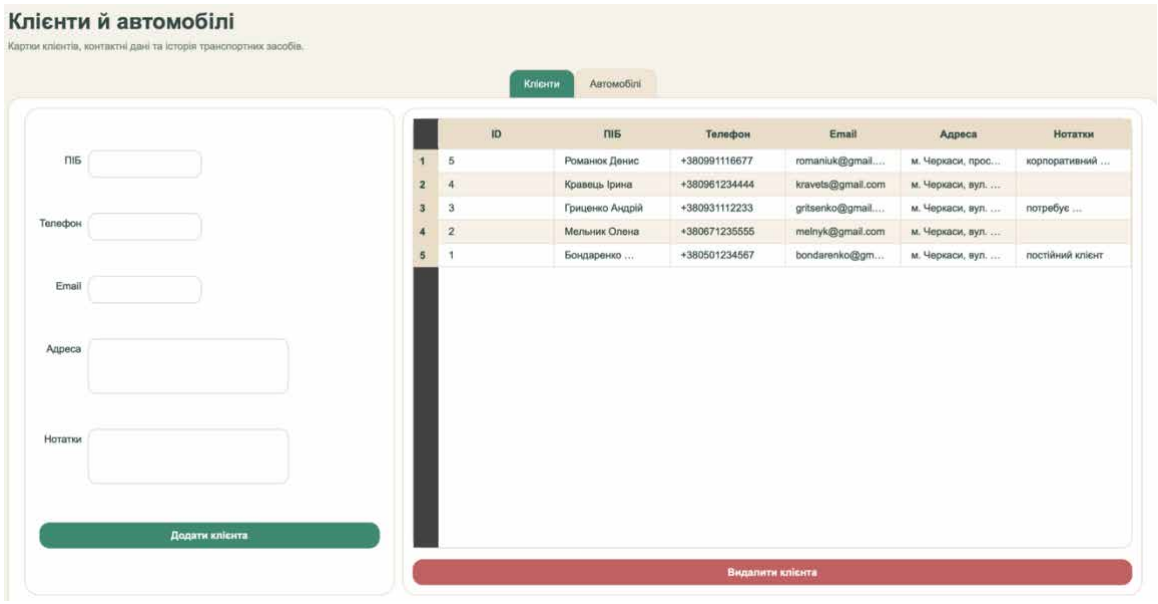

Рис. 4.4. Модуль обліку клієнтів СТО

На рис. 4.5 подано інтерфейс обліку автомобілів. У цьому модулі користувач обирає клієнта, вводить VIN, державний номер, марку, модель, рік

випуску та пробіг автомобіля. Під час тестування перевірено правильність прив’язування автомобіля до вибраного клієнта, збереження запису в базі даних і відображення інформації в таблиці.

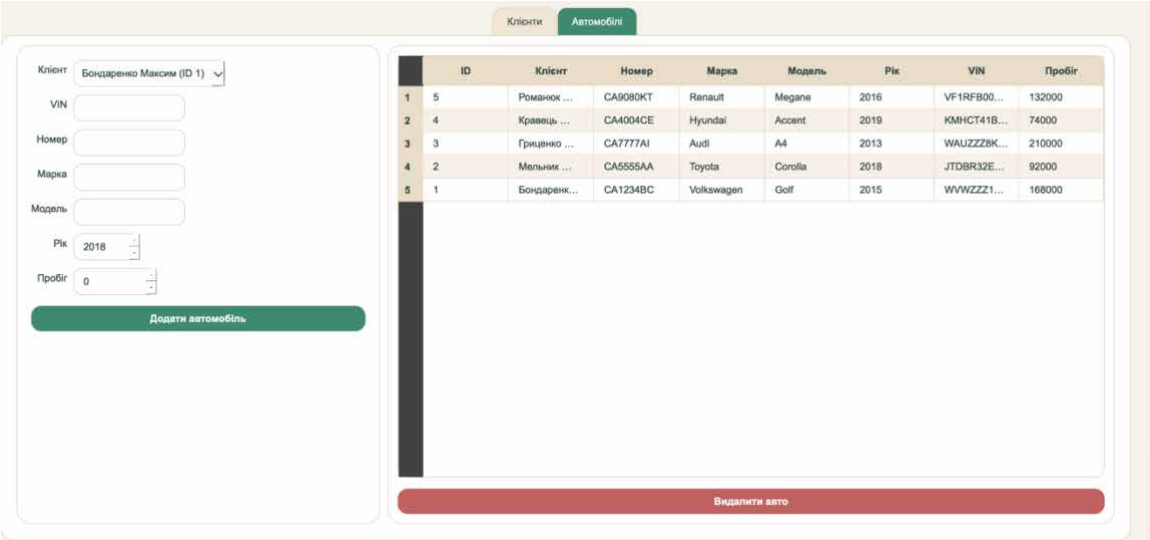

Рис. 4.5. Модуль обліку автомобілів клієнтів

Інтерфейс роботи із замовленнями-нарядами наведено на рис. 4.6. У модулі передбачено додавання послуги або запчастини до вибраного наряду. Під час перевірки встановлено, що після додавання послуги її назва, кількість, ціна та сума відображаються у складі замовлення. Також перевірено списання запчастин зі складу та автоматичне оновлення загальної вартості наряду.

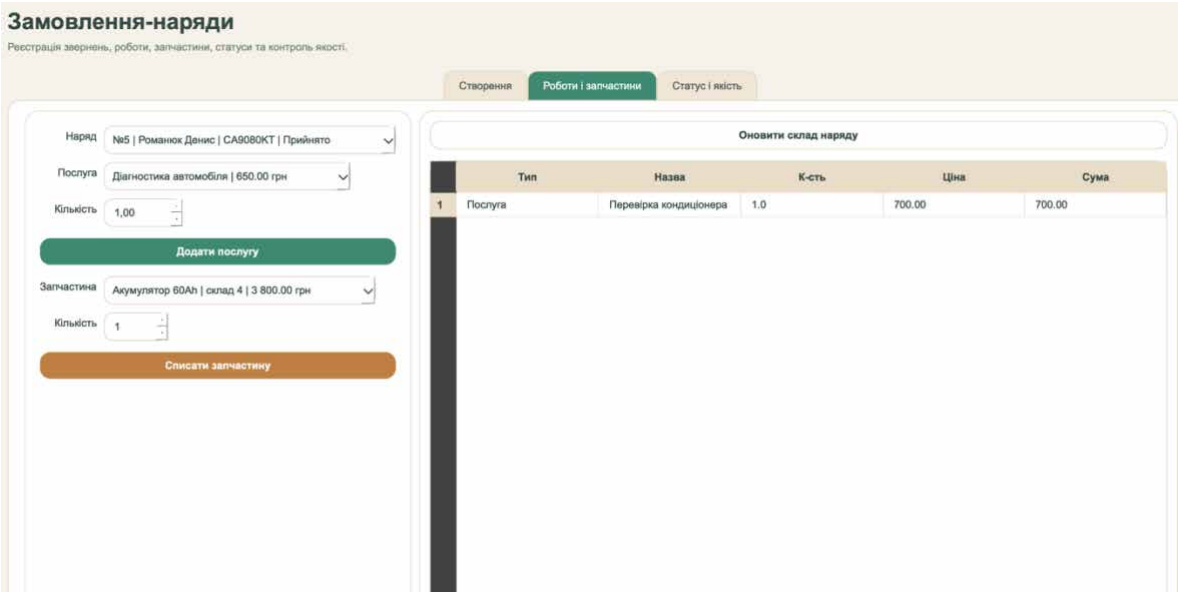

Рис. 4.6. Інтерфейс додавання робіт і запчастин до замовлення-наряду

Модуль складу і послуг, поданий на рис. 4.7, забезпечує ведення довідника сервісних операцій і складських позицій. У процесі тестування перевірено додавання нової послуги, введення категорії, ціни та норми часу. Таблична частина коректно відображає перелік наявних послуг, їх вартість і активність.

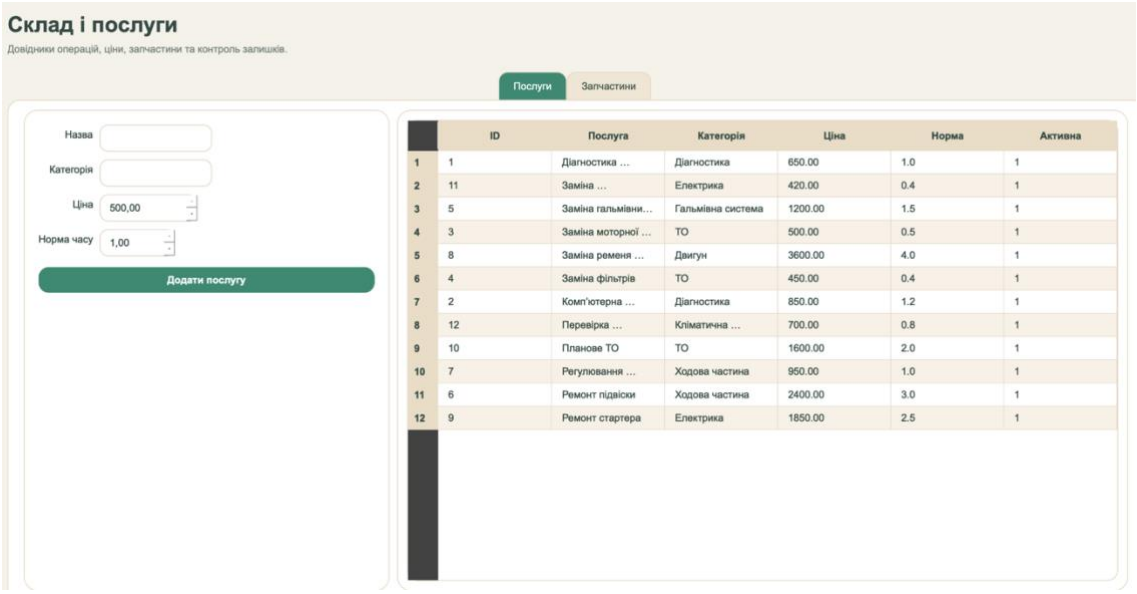

Рис. 4.7. Модуль довідника послуг і складського обліку

На рис. 4.8 наведено фінансовий модуль системи. Він використовується для формування рахунків, вибору наряду, реєстрації оплат і перегляду таблиць рахунків та платежів. Під час тестування перевірено, що рахунок створюється на основі суми замовлення, а після внесення платежу змінюється статус оплати.

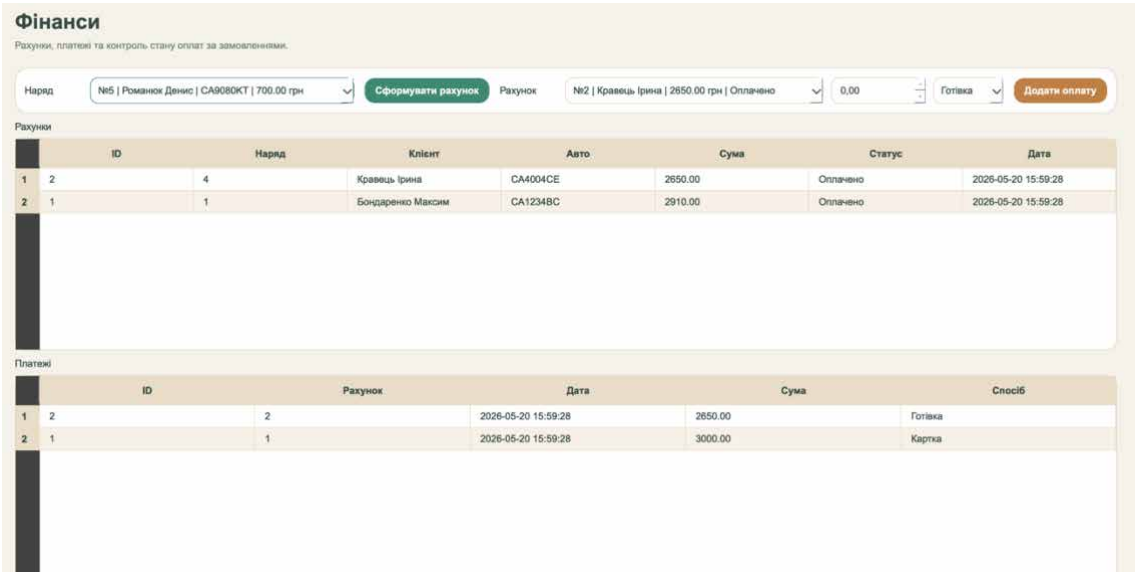

Рис. 4.8. Інтерфейс фінансового модуля системи

Інтерфейс KPI-аналітики подано на рис. 4.9. Модуль відображає кількість нарядів, завершені замовлення, середній чек, суму оплат, завантаження майстрів і фінансові надходження. Також передбачено експорт KPI у CSV-файл. Під час тестування встановлено, що аналітичні показники формуються на основі актуальних даних із таблиць замовлень, працівників, рахунків і платежів.

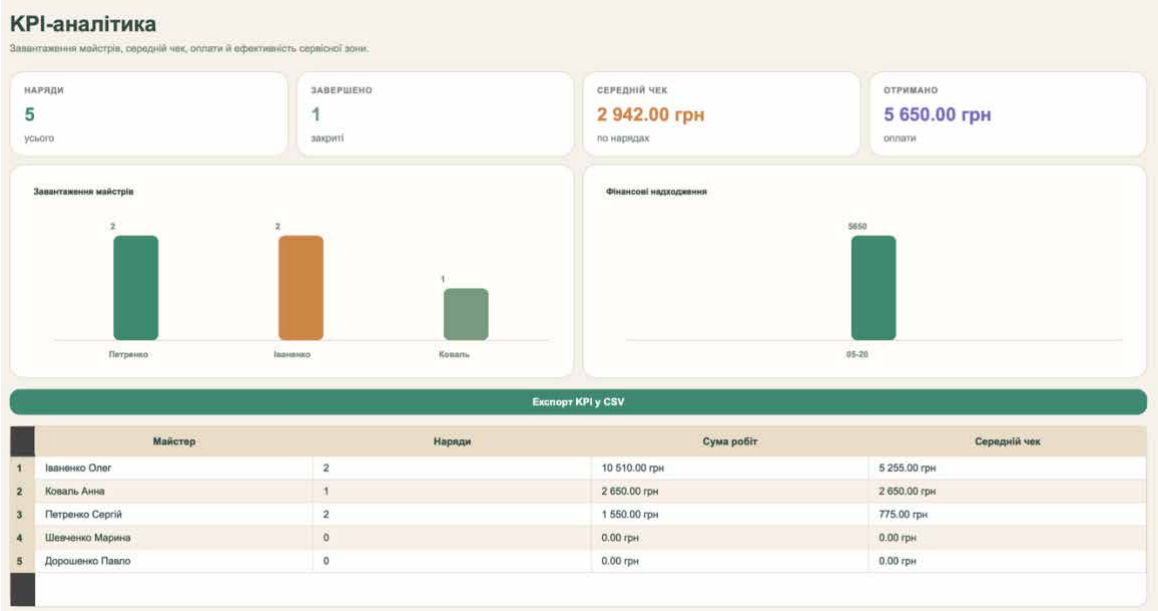

Рис. 4.9. Модуль KPI-аналітики облікової системи СТО

Таблиця 4.2 - Результати тестування інтерфейсних модулів системи

№	Інтерфейсний модуль	Перевірені дії	Результат
1	Вхід до системи	Введення email і пароля, перехід до головного вікна	Авторизація працює коректно
2	Реєстрація	Створення нового користувача з вибором ролі	Користувач додається до бази даних
3	Дашборд	Перегляд загальних показників і графіків	Дані відображаються відповідно до стану бази
4	Клієнти	Додавання нового клієнта та перегляд таблиці	Запис зберігається і відображається у списку
5	Автомобілі	Додавання автомобіля до вибраного клієнта	Автомобіль коректно прив’язується до клієнта
6	Замовлення-наряди	Додавання послуги та запчастини до наряду	Склад замовлення і сума оновлюються
7	Склад і послуги	Додавання нової послуги та перегляд довідника	Дані зберігаються у довіднику
8	Фінанси	Формування рахунку та внесення оплати	Рахунок і платіж створюються коректно
9	KPI-аналітика	Перегляд показників і графіків	Показники формуються на основі актуальних даних

Під час тестування інтерфейсних модулів також перевірено зрозумілість структури вікон, послідовність розміщення полів, доступність кнопок і коректність оновлення таблиць після виконання операцій. Верхня навігаційна панель забезпечує швидкий перехід між основними розділами системи: оглядом, клієнтами, нарядами, план-графіком, складом, фінансами, KPI-аналітикою та адмініструванням.

Результати тестування показали, що інтерфейсні модулі забезпечують виконання основних сценаріїв роботи СТО. Користувач може зареєструвати клієнта, додати автомобіль, створити замовлення-наряд, внести послуги та запчастини, сформувати рахунок, зареєструвати оплату й переглянути аналітичні показники. Отже, інтерфейс системи є придатним для практичного використання в межах навчального програмного прототипу.

4.3 Розгортання і впровадження системи, склад інсталяційного пакету

Розгортання облікової системи для станції технічного обслуговування передбачає встановлення настільного застосунку на робочу станцію користувача. Програмне забезпечення реалізовано мовою Python з використанням PyQt6, а дані зберігаються у локальній базі SQLite. Такий варіант розгортання є зручним для невеликого СТО, оскільки не потребує окремого веб-сервера, складного адміністрування або встановлення серверної СКБД.

Загальну структуру розгортання системи подано на рис. 4.10.

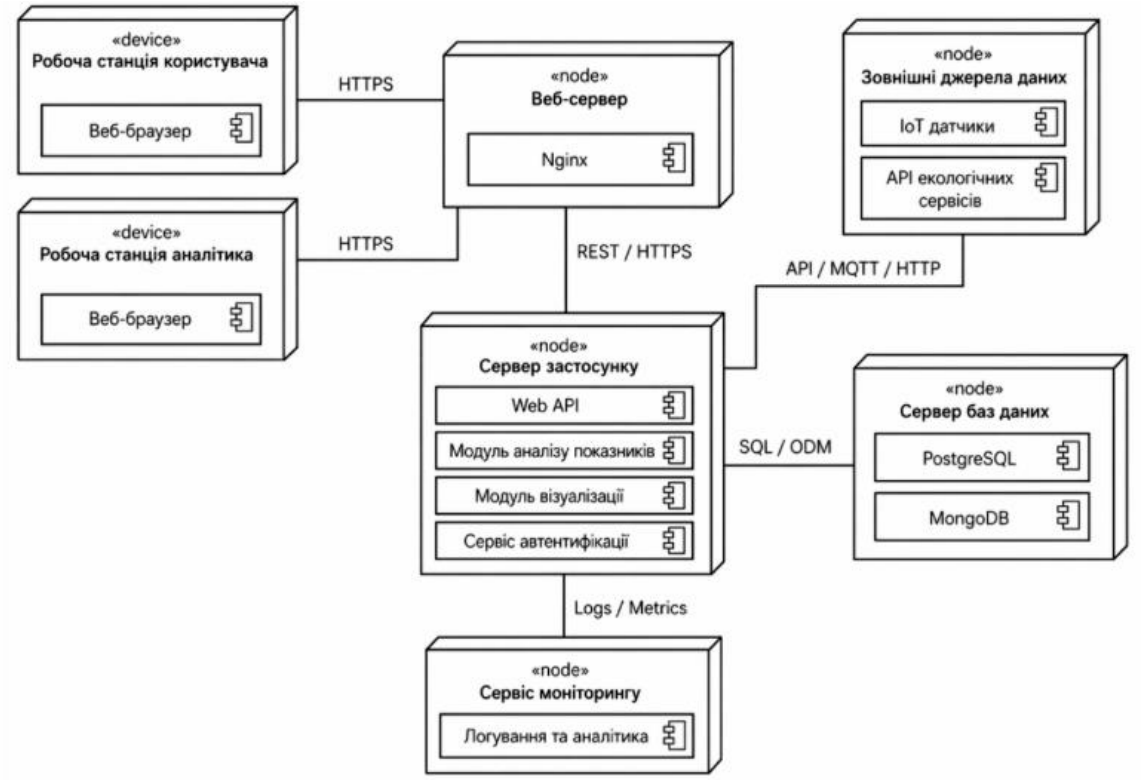

Рис. 4.10. Діаграма розгортання облікової системи СТО

Основним вузлом розгортання є робоча станція користувача. На ній запускається графічний застосунок STO Manager, який містить модулі авторизації, обліку клієнтів, автомобілів, замовлень-нарядів, послуг, запчастин, рахунків, оплат, план-графіка, аналітики та адміністрування. Взаємодія з базою даних виконується через вбудований модуль sqlite3.

База даних розміщується у файлі sto_manager.sqlite. У ній зберігаються всі основні дані системи: користувачі, клієнти, автомобілі, працівники, замовлення, послуги, запчастини, рахунки, платежі, записи план-графіка, сповіщення, контроль якості та журнал подій. Під час першого запуску база створюється автоматично, після чого система наповнюється початковими довідковими даними для перевірки працездатності.

Для запуску системи користувач розпаковує інсталяційний пакет, відкриває його у середовищі Visual Studio Code або через термінал, створює віртуальне середовище Python, встановлює залежності та запускає файл app.py. Приклад команд запуску наведено нижче.

```
cd sto_manager_pyqt6_fresh_ui_v2
python3 -m venv .venv
```

```
source .venv/bin/activate
pip install -r requirements.txt
python app.py
```

Після запуску відкривається вікно авторизації. Для перевірки роботи системи передбачено тестові облікові записи адміністратора, менеджера, майстра та бухгалтера. Це дає змогу перевірити роботу різних ролей користувачів і доступ до відповідних модулів.

Склад інсталяційного пакету наведено у таблиці 4.3.

Таблиця 4.3 - Склад інсталяційного пакету програмної системи

№	Елемент пакету	Призначення
1	app.py	Основний файл запуску програмного застосунку
2	requirements.txt	Перелік необхідних бібліотек Python
3	run_macos.sh	Скрипт швидкого запуску програми в macOS
4	run_windows.bat	Скрипт запуску програми в Windows
5	data	Каталог для зберігання SQLite-бази даних
6	sto_manager.sqlite	Файл бази даних, створюється автоматично після першого запуску
7	exports	Каталог для збереження CSV-звітів
8	README.md	Інструкція із запуску та короткий опис системи

Впровадження системи доцільно виконувати поетапно. На першому етапі встановлюється програмне середовище та перевіряється запуск застосунку. На другому етапі створюються або перевіряються облікові записи користувачів. На третьому етапі вносяться початкові довідники: працівники, послуги, запчастини та типові складські позиції. Після цього система може використовуватися для реєстрації клієнтів, автомобілів і замовлень-нарядів.

Під час впровадження перевіряється правильність створення бази даних, доступність основних модулів, робота авторизації, додавання клієнтів, створення замовлень, списання запчастин, формування рахунків і реєстрація оплат. Окремо перевіряється експорт KPI-звіту у CSV-файл.

Для забезпечення збереження даних рекомендовано періодично створювати резервну копію файлу sto_manager.sqlite. Оскільки база зберігається у вигляді одного файлу, резервне копіювання може виконуватися простим копіюванням цього файлу до окремої папки або на зовнішній носій.

Запропонований спосіб розгортання є достатнім для навчального програмного прототипу та невеликої станції технічного обслуговування. Він не потребує складної серверної інфраструктури, забезпечує швидкий запуск програми та дозволяє зберігати всі дані локально. У разі подальшого розвитку систему можна адаптувати до мережевого режиму роботи, винести базу даних на окремий сервер або реалізувати веб-версію для доступу з кількох робочих місць.

4.4 Висновки до четвертого розділу

У четвертому розділі виконано тестування та оцінювання ефективності облікової системи для станції технічного обслуговування. Перевірку проведено для основних програмних та інтерфейсних модулів, які забезпечують роботу з користувачами, клієнтами, автомобілями, замовленнями-нарядами, послугами, запчастинами, рахунками, оплатами, план-графіком і аналітичними показниками.

Під час тестування програмних модулів перевірено коректність авторизації та реєстрації користувачів, створення клієнтських карток, додавання автомобілів, формування замовлень-нарядів, зміни статусів ремонту, додавання послуг і списання запчастин. Окремо перевірено роботу фінансового модуля, який забезпечує створення рахунків, реєстрацію платежів і оновлення статусу оплати. Результати тестування підтвердили, що основні функції системи працюють відповідно до визначених вимог.

У процесі перевірки бази даних підтверджено правильність зв’язків між таблицями clients, vehicles, work_orders, services, spare_parts, invoices і payments. Система коректно зберігає дані, підтримує цілісність записів і не допускає виконання некоректних операцій, зокрема створення замовлення без автомобіля, списання запчастин без достатнього залишку або реєстрації платежу без рахунку.

Тестування інтерфейсних модулів показало, що користувач може послідовно виконувати основні сценарії роботи СТО: увійти до системи, зареєструвати клієнта, додати автомобіль, створити замовлення, внести роботи та запчастини, сформувати рахунок, зареєструвати оплату і переглянути аналітичні показники. Інтерфейс забезпечує швидкий перехід між модулями та відображає актуальні дані без потреби перезапуску програми.

Також перевірено модуль KPI-аналітики, який формує показники кількості замовлень, активних нарядів, завершених робіт, середнього чека, отриманих оплат і завантаження майстрів. Дані на графіках та у таблицях відповідають інформації, що зберігається в базі даних. Експорт аналітичного звіту у CSV-файл виконується коректно.

У межах розділу описано розгортання системи та склад інсталяційного пакету. Запропонований варіант запуску через Python, PyQt6 і SQLite є простим для використання, оскільки не потребує окремого сервера або складного налаштування. База даних створюється автоматично під час першого запуску, а всі основні файли програми розміщені в одному каталозі.

Отже, результати тестування підтвердили працездатність розробленої облікової системи СТО. Програмне забезпечення забезпечує повний цикл обліку сервісних процесів: від реєстрації клієнта та автомобіля до закриття замовлення, формування рахунку й аналізу показників роботи. Система може бути використана як навчальний програмний прототип і як основа для подальшого розширення функціональності.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було спроєктовано та реалізовано комплексне програмне забезпечення облікової системи для станції технічного обслуговування. Розроблена система призначена для автоматизації основних процесів СТО, зокрема обліку клієнтів, транспортних засобів, замовлень-нарядів, сервісних робіт, запчастин, рахунків, оплат і аналітичних показників діяльності.

У першому розділі було досліджено предметну область діяльності станції технічного обслуговування. Визначено, що традиційне ведення обліку за допомогою паперових журналів або розрізнених електронних таблиць ускладнює контроль замовлень, облік виконаних робіт, списання запчастин і формування фінансової звітності. Обґрунтовано доцільність створення програмної системи, яка забезпечує централізоване зберігання даних і підтримує повний цикл обслуговування клієнта.

Проведено аналіз існуючих рішень для автоматизації роботи автосервісів і сервісних підприємств. Установлено, що наявні програмні продукти мають широкий функціонал, проте часто є складними для впровадження, потребують налаштування або орієнтовані на великі підприємства. Це підтвердило актуальність розроблення навчального програмного прототипу, адаптованого до потреб невеликої станції технічного обслуговування.

У процесі моделювання предметної області визначено основні сутності системи: клієнт, автомобіль, працівник, замовлення-наряд, послуга, запчастина, рахунок і платіж. Побудовано логічну модель даних, UML-діаграму класів, діаграми кооперації, компонентів і пакетів. Розроблені моделі дали змогу формалізувати структуру програмного забезпечення та визначити зв’язки між основними модулями системи.

У другому розділі виконано проєктування інформаційного та програмного забезпечення системи. Логічна модель даних була побудована з урахуванням принципів нормалізації, що дозволило уникнути дублювання

інформації та забезпечити цілісність даних. У моделі передбачено окреме зберігання клієнтів, автомобілів, працівників, послуг, запчастин, замовлень, рахунків і оплат. Для зв’язків між замовленнями, послугами та запчастинами використано проміжні таблиці, що забезпечує коректне представлення складних зв’язків.

У третьому розділі обґрунтовано вибір технологічних засобів реалізації. Для створення програмного забезпечення використано мову Python, бібліотеку PyQt6 для побудови графічного інтерфейсу та SQLite для локального зберігання даних. Такий стек є доцільним для навчального прототипу, оскільки забезпечує простий запуск, не потребує окремого сервера та дає змогу реалізувати повноцінний настільний застосунок.

Практично реалізовано базу даних системи. У ній створено таблиці для зберігання користувачів, клієнтів, автомобілів, працівників, замовлень-нарядів, послуг, запчастин, рахунків, платежів, план-графіка, контрольних перевірок якості та журналу подій. Реалізовано зв’язки між таблицями, первинні та зовнішні ключі, а також механізми збереження фактичної вартості послуг і запчастин на момент додавання до замовлення.

Розроблено алгоритми роботи основних програмних модулів. Описано послідовність реєстрації клієнта та автомобіля, створення замовлення-наряду, додавання послуг і запчастин, формування рахунку, реєстрації оплати та закриття замовлення. Алгоритми передбачають перевірку вхідних даних, контроль складських залишків, перерахунок суми замовлення та оновлення статусів.

У четвертому розділі проведено тестування програмних та інтерфейсних модулів системи. Перевірено авторизацію і реєстрацію користувачів, додавання клієнтів, створення автомобілів, оформлення замовлень-нарядів, додавання робіт і запчастин, формування рахунків, внесення оплат, перегляд аналітики та експорт звітів. Результати тестування підтвердили працездатність основних функцій системи.

Також перевірено коректність роботи інтерфейсу користувача. Програма забезпечує доступ до основних розділів через навігаційну панель, коректно відображає таблиці, форми введення, графіки та інформаційні показники. Дані оновлюються після виконання операцій без необхідності перезапуску застосунку.

Розроблена система забезпечує автоматизацію повного циклу роботи СТО: від реєстрації клієнта та автомобіля до створення замовлення, фіксації виконаних робіт, списання запчастин, формування рахунку, реєстрації оплати й аналізу результатів діяльності. Це дозволяє зменшити кількість ручних операцій, підвищити точність обліку та спростити контроль за виконанням сервісних процесів.

Практична цінність роботи полягає в тому, що створений програмний продукт може бути використаний як навчальний прототип для демонстрації автоматизації діяльності СТО. Надалі систему можна розширити шляхом додавання мережевого режиму роботи, резервного копіювання бази даних, інтеграції з SMS або email-сповіщеннями, фіскальним модулем, вебкабінетом клієнта та розширеною аналітикою ефективності роботи сервісу.

Отже, мету кваліфікаційної роботи досягнуто. Спроєктовано, реалізовано та протестовано програмну систему, яка забезпечує облік клієнтів, автомобілів, замовлень, робіт, запчастин, рахунків, оплат і аналітичних показників станції технічного обслуговування. Розроблене рішення відповідає поставленим функціональним вимогам і може бути використане як основа для подальшого розвитку інформаційної системи СТО.

ДОДАТОК А

Фрагмент програмного коду створення бази даних облікової системи СТО

```
# database.py
import sqlite3
from pathlib import Path
from datetime import datetime
import hashlib

DB_PATH = Path("data") / "sto_manager.sqlite"


def get_connection():
    DB_PATH.parent.mkdir(parents=True, exist_ok=True)
    connection = sqlite3.connect(DB_PATH)
    connection.row_factory = sqlite3.Row
    connection.execute("PRAGMA foreign_keys = ON;")
    return connection


def hash_password(password):
    return hashlib.sha256(password.encode("utf-8")).hexdigest()


def init_database():
    connection = get_connection()
    cursor = connection.cursor()

    cursor.executescript("""
    CREATE TABLE IF NOT EXISTS users (
        user_id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT NOT NULL UNIQUE,
        password_hash TEXT NOT NULL,
        role TEXT NOT NULL CHECK(role IN ('admin', 'manager', 'master',
'accountant')),
        full_name TEXT NOT NULL,
        created_at TEXT NOT NULL
    );

    CREATE TABLE IF NOT EXISTS clients (
        client_id INTEGER PRIMARY KEY AUTOINCREMENT,
        full_name TEXT NOT NULL,
        phone TEXT NOT NULL UNIQUE,
        email TEXT,
        address TEXT,
        created_at TEXT NOT NULL
    );

    CREATE TABLE IF NOT EXISTS vehicles (
        vehicle_id INTEGER PRIMARY KEY AUTOINCREMENT,
        client_id INTEGER NOT NULL,
        vin TEXT NOT NULL UNIQUE,
        license_plate TEXT NOT NULL UNIQUE,
        make TEXT NOT NULL,
        model TEXT NOT NULL,
        production_year INTEGER NOT NULL,
        mileage INTEGER DEFAULT 0,
        FOREIGN KEY (client_id) REFERENCES clients(client_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE
```

```
);

CREATE TABLE IF NOT EXISTS employees (
    employee_id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name TEXT NOT NULL,
    position TEXT NOT NULL,
    phone TEXT,
    specialization TEXT,
    is_active INTEGER NOT NULL DEFAULT 1
);

CREATE TABLE IF NOT EXISTS services (
    service_id INTEGER PRIMARY KEY AUTOINCREMENT,
    service_name TEXT NOT NULL UNIQUE,
    duration_minutes INTEGER NOT NULL,
    price REAL NOT NULL,
    is_active INTEGER NOT NULL DEFAULT 1
);

CREATE TABLE IF NOT EXISTS spare_parts (
    part_id INTEGER PRIMARY KEY AUTOINCREMENT,
    part_name TEXT NOT NULL,
    manufacturer TEXT,
    unit_price REAL NOT NULL,
    stock_qty INTEGER NOT NULL DEFAULT 0,
    min_qty INTEGER NOT NULL DEFAULT 1
);

CREATE TABLE IF NOT EXISTS work_orders (
    order_id INTEGER PRIMARY KEY AUTOINCREMENT,
    client_id INTEGER NOT NULL,
    vehicle_id INTEGER NOT NULL,
    employee_id INTEGER,
    opened_at TEXT NOT NULL,
    closed_at TEXT,
    complaint TEXT NOT NULL,
    diagnostic_result TEXT,
    status TEXT NOT NULL DEFAULT 'прийнято',
    total_amount REAL NOT NULL DEFAULT 0,
    FOREIGN KEY (client_id) REFERENCES clients(client_id)
        ON UPDATE CASCADE
ON DELETE RESTRICT,
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
        ON UPDATE CASCADE
        ON DELETE SET NULL
);

CREATE TABLE IF NOT EXISTS work_order_services (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    order_id INTEGER NOT NULL,
    service_id INTEGER NOT NULL,
    quantity INTEGER NOT NULL DEFAULT 1,
    service_price REAL NOT NULL,
    FOREIGN KEY (order_id) REFERENCES work_orders(order_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (service_id) REFERENCES services(service_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);
```

```
    CREATE TABLE IF NOT EXISTS work_order_parts (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        order_id INTEGER NOT NULL,
        part_id INTEGER NOT NULL,
        quantity INTEGER NOT NULL,
        part_price REAL NOT NULL,
        FOREIGN KEY (order_id) REFERENCES work_orders(order_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE,
        FOREIGN KEY (part_id) REFERENCES spare_parts(part_id)
            ON UPDATE CASCADE
            ON DELETE RESTRICT
    );

    CREATE TABLE IF NOT EXISTS invoices (
        invoice_id INTEGER PRIMARY KEY AUTOINCREMENT,
        order_id INTEGER NOT NULL UNIQUE,
        invoice_date TEXT NOT NULL,
        total_amount REAL NOT NULL,
        payment_status TEXT NOT NULL DEFAULT 'не оплачено',
        FOREIGN KEY (order_id) REFERENCES work_orders(order_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE
    );

    CREATE TABLE IF NOT EXISTS payments (
        payment_id INTEGER PRIMARY KEY AUTOINCREMENT,
        invoice_id INTEGER NOT NULL,
        payment_date TEXT NOT NULL,
        amount REAL NOT NULL,
        payment_method TEXT NOT NULL,
        FOREIGN KEY (invoice_id) REFERENCES invoices(invoice_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE
    );

    CREATE TABLE IF NOT EXISTS audit_log (
        log_id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER,
        action TEXT NOT NULL,
        entity_name TEXT NOT NULL,
        entity_id INTEGER,
        created_at TEXT NOT NULL,
        FOREIGN KEY (user_id) REFERENCES users(user_id)
            ON UPDATE CASCADE
            ON DELETE SET NULL
    );

    CREATE INDEX IF NOT EXISTS idx_clients_phone ON clients(phone);
    CREATE INDEX IF NOT EXISTS idx_vehicles_vin ON vehicles(vin);
    CREATE INDEX IF NOT EXISTS idx_work_orders_status ON work_orders(status);
    CREATE INDEX IF NOT EXISTS idx_work_orders_vehicle ON
work_orders(vehicle_id);
    CREATE INDEX IF NOT EXISTS idx_invoice_status ON invoices(payment_status);
    """)

    connection.commit()
    connection.close()


def seed_database():
    connection = get_connection()
```

```
    cursor = connection.cursor()

    cursor.execute("SELECT COUNT(*) AS count_users FROM users")
    if cursor.fetchone()["count_users"] > 0:
        connection.close()
        return

    now = datetime.now().isoformat(timespec="seconds")

    users = [
        ("admin", hash_password("admin123"), "admin", "Адміністратор системи",
now),
        ("manager", hash_password("manager123"), "manager", "Менеджер СТО",
now),
        ("master", hash_password("master123"), "master", "Майстер СТО", now),
        ("accountant", hash_password("accountant123"), "accountant",
"Бухгалтер", now)
    ]

    cursor.executemany("""
        INSERT INTO users (username, password_hash, role, full_name, created_at)
        VALUES (?, ?, ?, ?, ?)
    """, users)

    cursor.execute("""
        INSERT INTO clients (full_name, phone, email, address, created_at)
        VALUES (?, ?, ?, ?, ?)
    """, ("Іваненко Петро Сергійович", "+380501112233", "ivanenko@example.com",
          "м. Київ, вул. Центральна, 15", now))

    cursor.execute("""
        INSERT INTO vehicles
        (client_id, vin, license_plate, make, model, production_year, mileage)
        VALUES (?, ?, ?, ?, ?, ?, ?)
    """, (1, "WVWZZZ1KZ8W123456", "AA1234KT", "Volkswagen", "Golf", 2018,
156000))

    services = [
        ("Комп’ютерна діагностика", 40, 600.00, 1),
        ("Заміна моторної оливи", 60, 800.00, 1),
        ("Заміна гальмівних колодок", 90, 1200.00, 1),
        ("Діагностика ходової частини", 50, 700.00, 1)
    ]

    cursor.executemany("""
        INSERT INTO services (service_name, duration_minutes, price, is_active)
        VALUES (?, ?, ?, ?)
    """, services)

    spare_parts = [
        ("Фільтр масляний", "Bosch", 320.00, 15, 3),
        ("Олива моторна 5W-30", "Castrol", 420.00, 30, 5),
        ("Колодки гальмівні передні", "TRW", 1850.00, 8, 2),
        ("Фільтр повітряний", "Mann", 480.00, 10, 2)
    ]

    cursor.executemany("""
        INSERT INTO spare_parts (part_name, manufacturer, unit_price, stock_qty,
min_qty)
        VALUES (?, ?, ?, ?, ?)
    """, spare_parts)

    cursor.execute("""
```

```
        INSERT INTO employees (full_name, position, phone, specialization,
is_active)
        VALUES (?, ?, ?, ?, ?)
    """, ("Коваленко Андрій Миколайович", "Майстер", "+380671112233",
          "Діагностика та ремонт двигуна", 1))

    connection.commit()
    connection.close()


if __name__ == "__main__":
    init_database()
    seed_database()
    print("Базу даних облікової системи СТО створено.")
```

ДОДАТОК Б

Фрагмент програмного коду сервісного шару оброблення замовлень-нарядів

```
# order_service.py
from datetime import datetime
from database import get_connection


class WorkOrderService:
    def create_order(self, client_id, vehicle_id, employee_id, complaint):
        if not complaint.strip():
            raise ValueError("Опис звернення клієнта не може бути порожнім.")

        connection = get_connection()
        cursor = connection.cursor()

        cursor.execute("""
            INSERT INTO work_orders
            (client_id, vehicle_id, employee_id, opened_at, complaint, status,
total_amount)
            VALUES (?, ?, ?, ?, ?, ?, ?)
        """, (
            client_id,
            vehicle_id,
            employee_id,
            datetime.now().isoformat(timespec="seconds"),
            complaint.strip(),
            "прийнято",
            0.00
        ))

        order_id = cursor.lastrowid
        self._write_log(cursor, None, "Створено замовлення-наряд",
"work_orders", order_id)

        connection.commit()
        connection.close()

        return order_id

    def add_service_to_order(self, order_id, service_id, quantity=1):
        if quantity <= 0:
            raise ValueError("Кількість послуг має бути більшою за нуль.")

        connection = get_connection()
        cursor = connection.cursor()

        service = cursor.execute("""
            SELECT service_id, price
            FROM services
            WHERE service_id = ? AND is_active = 1
        """, (service_id,)).fetchone()

        if service is None:
            connection.close()
            raise ValueError("Послугу не знайдено або вона неактивна.")

        cursor.execute("""
            INSERT INTO work_order_services
```

```
            (order_id, service_id, quantity, service_price)
            VALUES (?, ?, ?, ?)
        """, (order_id, service_id, quantity, service["price"]))

        self._recalculate_order_total(cursor, order_id)
        self._write_log(cursor, None, "Додано послугу до замовлення",
"work_orders", order_id)

        connection.commit()
        connection.close()

    def add_part_to_order(self, order_id, part_id, quantity):
        if quantity <= 0:
            raise ValueError("Кількість запчастин має бути більшою за нуль.")

        connection = get_connection()
        cursor = connection.cursor()

        part = cursor.execute("""
            SELECT part_id, unit_price, stock_qty
            FROM spare_parts
            WHERE part_id = ?
        """, (part_id,)).fetchone()

        if part is None:
            connection.close()
            raise ValueError("Запчастину не знайдено.")

        if part["stock_qty"] < quantity:
            connection.close()
            raise ValueError("Недостатня кількість запчастин на складі.")

        cursor.execute("""
            INSERT INTO work_order_parts
            (order_id, part_id, quantity, part_price)
            VALUES (?, ?, ?, ?)
        """, (order_id, part_id, quantity, part["unit_price"]))

        cursor.execute("""
            UPDATE spare_parts
            SET stock_qty = stock_qty - ?
            WHERE part_id = ?
        """, (quantity, part_id))

        self._recalculate_order_total(cursor, order_id)
        self._write_log(cursor, None, "Списано запчастину на замовлення",
"work_orders", order_id)

        connection.commit()
        connection.close()

    def update_order_status(self, order_id, status, diagnostic_result=None):
        allowed_statuses = [
            "прийнято",
            "в роботі",
            "очікує запчастини",
            "завершено",
            "скасовано"
        ]

        if status not in allowed_statuses:
            raise ValueError("Некоректний статус замовлення.")
```

```
        connection = get_connection()
        cursor = connection.cursor()

        closed_at = None
        if status == "завершено":
            closed_at = datetime.now().isoformat(timespec="seconds")

        cursor.execute("""
            UPDATE work_orders
            SET status = ?,
                diagnostic_result = COALESCE(?, diagnostic_result),
                closed_at = COALESCE(?, closed_at)
            WHERE order_id = ?
        """, (status, diagnostic_result, closed_at, order_id))

        self._write_log(cursor, None, "Оновлено статус замовлення",
"work_orders", order_id)

        connection.commit()
        connection.close()

    def create_invoice(self, order_id):
        connection = get_connection()
        cursor = connection.cursor()

        order = cursor.execute("""
            SELECT order_id, total_amount
            FROM work_orders
            WHERE order_id = ?
        """, (order_id,)).fetchone()

        if order is None:
            connection.close()
            raise ValueError("Замовлення-наряд не знайдено.")

        existing_invoice = cursor.execute("""
            SELECT invoice_id
            FROM invoices
            WHERE order_id = ?
        """, (order_id,)).fetchone()

        if existing_invoice is not None:
            connection.close()
            return existing_invoice["invoice_id"]

        cursor.execute("""
            INSERT INTO invoices
            (order_id, invoice_date, total_amount, payment_status)
            VALUES (?, ?, ?, ?)
        """, (
            order_id,
            datetime.now().isoformat(timespec="seconds"),
            order["total_amount"],
            "не оплачено"
        ))

        invoice_id = cursor.lastrowid
        self._write_log(cursor, None, "Сформовано рахунок", "invoices",
invoice_id)

        connection.commit()
        connection.close()
```

```
        return invoice_id

    def register_payment(self, invoice_id, amount, payment_method):
        if amount <= 0:
            raise ValueError("Сума оплати має бути більшою за нуль.")

        connection = get_connection()
        cursor = connection.cursor()

        invoice = cursor.execute("""
            SELECT invoice_id, total_amount
            FROM invoices
            WHERE invoice_id = ?
        """, (invoice_id,)).fetchone()

        if invoice is None:
            connection.close()
            raise ValueError("Рахунок не знайдено.")

        cursor.execute("""
            INSERT INTO payments
            (invoice_id, payment_date, amount, payment_method)
            VALUES (?, ?, ?, ?)
        """, (
            invoice_id,
            datetime.now().isoformat(timespec="seconds"),
            amount,
            payment_method
        ))

        paid_sum = cursor.execute("""
            SELECT COALESCE(SUM(amount), 0) AS paid_sum
            FROM payments
            WHERE invoice_id = ?
        """, (invoice_id,)).fetchone()["paid_sum"]

        if paid_sum >= invoice["total_amount"]:
            payment_status = "оплачено"
        elif paid_sum > 0:
            payment_status = "частково оплачено"
        else:
            payment_status = "не оплачено"

        cursor.execute("""
            UPDATE invoices
            SET payment_status = ?
            WHERE invoice_id = ?
        """, (payment_status, invoice_id))

        self._write_log(cursor, None, "Зареєстровано оплату", "payments",
invoice_id)

        connection.commit()
        connection.close()

    def get_order_summary(self, order_id):
        connection = get_connection()
        cursor = connection.cursor()

        order = cursor.execute("""
            SELECT wo.order_id, c.full_name AS client_name, v.license_plate,
                   v.make, v.model, e.full_name AS employee_name,
                   wo.opened_at, wo.status, wo.total_amount, wo.complaint
```

```
            FROM work_orders wo
            JOIN clients c ON c.client_id = wo.client_id
            JOIN vehicles v ON v.vehicle_id = wo.vehicle_id
            LEFT JOIN employees e ON e.employee_id = wo.employee_id
            WHERE wo.order_id = ?
        """, (order_id,)).fetchone()

        services = cursor.execute("""
            SELECT s.service_name, wos.quantity, wos.service_price
            FROM work_order_services wos
            JOIN services s ON s.service_id = wos.service_id
            WHERE wos.order_id = ?
        """, (order_id,)).fetchall()

        parts = cursor.execute("""
            SELECT sp.part_name, wop.quantity, wop.part_price
            FROM work_order_parts wop
            JOIN spare_parts sp ON sp.part_id = wop.part_id
            WHERE wop.order_id = ?
        """, (order_id,)).fetchall()

        connection.close()

        return {
            "order": dict(order) if order else None,
            "services": [dict(row) for row in services],
            "parts": [dict(row) for row in parts]
        }

    def _recalculate_order_total(self, cursor, order_id):
        services_total = cursor.execute("""
            SELECT COALESCE(SUM(quantity * service_price), 0) AS total
            FROM work_order_services
            WHERE order_id = ?
        """, (order_id,)).fetchone()["total"]

        parts_total = cursor.execute("""
            SELECT COALESCE(SUM(quantity * part_price), 0) AS total
            FROM work_order_parts
            WHERE order_id = ?
        """, (order_id,)).fetchone()["total"]

        total_amount = services_total + parts_total

        cursor.execute("""
            UPDATE work_orders
            SET total_amount = ?
            WHERE order_id = ?
        """, (total_amount, order_id))

    def _write_log(self, cursor, user_id, action, entity_name, entity_id):
        cursor.execute("""
            INSERT INTO audit_log
            (user_id, action, entity_name, entity_id, created_at)
            VALUES (?, ?, ?, ?, ?)
        """, (
            user_id,
            action,
            entity_name,
            entity_id,
            datetime.now().isoformat(timespec="seconds")
        ))
```

ДОДАТОК В

Фрагмент програмного коду головного вікна PyQt6-застосунку

```
# app.py
import sys
from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QMainWindow,
    QWidget,
    QVBoxLayout,
    QHBoxLayout,
    QLabel,
    QLineEdit,
    QPushButton,
    QTableWidget,
    QTableWidgetItem,
    QTabWidget,
    QMessageBox,
    QHeaderView,
    QComboBox,
    QTextEdit
)

from database import init_database, seed_database, get_connection, hash_password
from order_service import WorkOrderService


class LoginWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("STO Manager - авторизація")
        self.setFixedSize(420, 250)

        layout = QVBoxLayout()

        title = QLabel("Облікова система станції технічного обслуговування")
        title.setAlignment(Qt.AlignmentFlag.AlignCenter)
        title.setObjectName("title")

        self.login_input = QLineEdit()
        self.login_input.setPlaceholderText("Логін")

        self.password_input = QLineEdit()
        self.password_input.setPlaceholderText("Пароль")
        self.password_input.setEchoMode(QLineEdit.EchoMode.Password)

        login_button = QPushButton("Увійти")
        login_button.clicked.connect(self.login)

        hint = QLabel("Тестовий вхід: admin / admin123")
        hint.setAlignment(Qt.AlignmentFlag.AlignCenter)
        hint.setObjectName("hint")

        layout.addWidget(title)
        layout.addSpacing(12)
        layout.addWidget(self.login_input)
        layout.addWidget(self.password_input)
        layout.addWidget(login_button)
```

```
        layout.addWidget(hint)

        self.setLayout(layout)

    def login(self):
        username = self.login_input.text().strip()
        password = self.password_input.text().strip()

        connection = get_connection()
        cursor = connection.cursor()

        user = cursor.execute("""
            SELECT user_id, username, role, full_name
            FROM users
            WHERE username = ? AND password_hash = ?
        """, (username, hash_password(password))).fetchone()

        connection.close()

        if user is None:
            QMessageBox.warning(self, "Помилка", "Невірний логін або пароль.")
            return

        self.main_window = MainWindow(dict(user))
        self.main_window.show()
        self.close()


class MainWindow(QMainWindow):
    def __init__(self, user):
        super().__init__()
        self.user = user
        self.order_service = WorkOrderService()

        self.setWindowTitle("STO Manager - облікова система СТО")
        self.resize(1180, 760)

        self.tabs = QTabWidget()
        self.setCentralWidget(self.tabs)

        self.dashboard_tab = QWidget()
        self.clients_tab = QWidget()
        self.orders_tab = QWidget()
        self.parts_tab = QWidget()
        self.finance_tab = QWidget()

        self.tabs.addTab(self.dashboard_tab, "Дашборд")
        self.tabs.addTab(self.clients_tab, "Клієнти та авто")
        self.tabs.addTab(self.orders_tab, "Замовлення-наряди")
        self.tabs.addTab(self.parts_tab, "Склад")
        self.tabs.addTab(self.finance_tab, "Фінанси")

        self.build_dashboard_tab()
        self.build_clients_tab()
        self.build_orders_tab()
        self.build_parts_tab()
        self.build_finance_tab()

        self.refresh_all()

    def build_dashboard_tab(self):
        layout = QVBoxLayout()
```

```
        title = QLabel(f"Користувач: {self.user['full_name']} | Роль:
{self.user['role']}")
        title.setObjectName("sectionTitle")

        cards = QHBoxLayout()

        self.clients_count = QLabel()
        self.vehicles_count = QLabel()
        self.active_orders_count = QLabel()
        self.revenue_sum = QLabel()

        for card in [self.clients_count, self.vehicles_count,
self.active_orders_count, self.revenue_sum]:
            card.setObjectName("card")
            card.setAlignment(Qt.AlignmentFlag.AlignCenter)
            cards.addWidget(card)

        refresh_button = QPushButton("Оновити показники")
        refresh_button.clicked.connect(self.refresh_all)

        layout.addWidget(title)
        layout.addLayout(cards)
        layout.addWidget(refresh_button)
        layout.addStretch()

        self.dashboard_tab.setLayout(layout)

    def build_clients_tab(self):
        layout = QVBoxLayout()

        title = QLabel("Клієнти та транспортні засоби")
        title.setObjectName("sectionTitle")

        self.clients_table = QTableWidget()
        self.clients_table.setColumnCount(6)
        self.clients_table.setHorizontalHeaderLabels([
            "Клієнт",
            "Телефон",
            "Email",
            "Авто",
            "Номер",
            "VIN"
        ])

self.clients_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMod
e.Stretch)

        layout.addWidget(title)
        layout.addWidget(self.clients_table)

        self.clients_tab.setLayout(layout)

    def build_orders_tab(self):
        layout = QVBoxLayout()

        title = QLabel("Замовлення-наряди")
        title.setObjectName("sectionTitle")

        form_layout = QHBoxLayout()

        self.client_combo = QComboBox()
        self.vehicle_combo = QComboBox()
        self.employee_combo = QComboBox()
```

```
        self.complaint_input = QTextEdit()
        self.complaint_input.setPlaceholderText("Опис звернення клієнта")
        self.complaint_input.setFixedHeight(70)

        create_button = QPushButton("Створити замовлення")
        create_button.clicked.connect(self.create_order)

        form_layout.addWidget(self.client_combo)
        form_layout.addWidget(self.vehicle_combo)
        form_layout.addWidget(self.employee_combo)
        form_layout.addWidget(create_button)

        self.orders_table = QTableWidget()
        self.orders_table.setColumnCount(8)
        self.orders_table.setHorizontalHeaderLabels([
            "ID",
            "Клієнт",
            "Авто",
            "Майстер",
            "Дата",
            "Скарга",
            "Статус",
            "Сума"
        ])

self.orders_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode
.Stretch)

        layout.addWidget(title)
        layout.addLayout(form_layout)
        layout.addWidget(self.complaint_input)
        layout.addWidget(self.orders_table)

        self.orders_tab.setLayout(layout)

    def build_parts_tab(self):
        layout = QVBoxLayout()

        title = QLabel("Склад запасних частин")
        title.setObjectName("sectionTitle")

        self.parts_table = QTableWidget()
        self.parts_table.setColumnCount(5)
        self.parts_table.setHorizontalHeaderLabels([
            "Назва",
            "Виробник",
            "Ціна",
            "Залишок",
            "Мін. залишок"
        ])

self.parts_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.
Stretch)

        layout.addWidget(title)
        layout.addWidget(self.parts_table)

        self.parts_tab.setLayout(layout)

    def build_finance_tab(self):
        layout = QVBoxLayout()
```

```
        title = QLabel("Рахунки та оплати")
        title.setObjectName("sectionTitle")

        self.finance_table = QTableWidget()
        self.finance_table.setColumnCount(6)
        self.finance_table.setHorizontalHeaderLabels([
            "Рахунок",
            "Замовлення",
            "Дата",
            "Сума",
            "Статус",
            "Оплачено"
        ])

self.finance_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMod
e.Stretch)

        layout.addWidget(title)
        layout.addWidget(self.finance_table)

        self.finance_tab.setLayout(layout)

    def create_order(self):
        client_id = self.client_combo.currentData()
        vehicle_id = self.vehicle_combo.currentData()
        employee_id = self.employee_combo.currentData()
        complaint = self.complaint_input.toPlainText().strip()

        if client_id is None or vehicle_id is None:
            QMessageBox.warning(self, "Помилка", "Оберіть клієнта та
автомобіль.")
            return

        try:
            order_id = self.order_service.create_order(
                client_id=client_id,
                vehicle_id=vehicle_id,
                employee_id=employee_id,
                complaint=complaint
            )

            QMessageBox.information(
                self,
                "Замовлення створено",
                f"Створено замовлення-наряд № {order_id}."
            )

            self.complaint_input.clear()
            self.refresh_all()

        except Exception as error:
            QMessageBox.critical(self, "Помилка", str(error))

    def refresh_all(self):
        self.refresh_dashboard()
        self.refresh_clients()
        self.refresh_orders()
        self.refresh_parts()
        self.refresh_finance()
        self.refresh_combos()

    def refresh_dashboard(self):
        connection = get_connection()
```

```
        cursor = connection.cursor()

        clients = cursor.execute("SELECT COUNT(*) AS value FROM
clients").fetchone()["value"]
        vehicles = cursor.execute("SELECT COUNT(*) AS value FROM
vehicles").fetchone()["value"]
        active_orders = cursor.execute("""
            SELECT COUNT(*) AS value
            FROM work_orders
            WHERE status != 'завершено' AND status != 'скасовано'
        """).fetchone()["value"]

        revenue = cursor.execute("""
            SELECT COALESCE(SUM(amount), 0) AS value
            FROM payments
        """).fetchone()["value"]

        connection.close()

        self.clients_count.setText(f"Клієнти\n{clients}")
        self.vehicles_count.setText(f"Автомобілі\n{vehicles}")
        self.active_orders_count.setText(f"Активні замовлення\n{active_orders}")
        self.revenue_sum.setText(f"Оплати\n{revenue:.2f} грн")

    def refresh_clients(self):
        connection = get_connection()
        cursor = connection.cursor()

        rows = cursor.execute("""
            SELECT c.full_name, c.phone, c.email,
                   v.make || ' ' || v.model AS car,
                   v.license_plate, v.vin
            FROM clients c
            LEFT JOIN vehicles v ON v.client_id = c.client_id
            ORDER BY c.full_name
        """).fetchall()

        connection.close()

        self.clients_table.setRowCount(len(rows))

        for row_index, row in enumerate(rows):
            values = [
                row["full_name"],
                row["phone"],
                row["email"] or "-",
                row["car"] or "-",
                row["license_plate"] or "-",
                row["vin"] or "-"
            ]

            for column_index, value in enumerate(values):
                self.clients_table.setItem(row_index, column_index,
QTableWidgetItem(str(value)))

    def refresh_orders(self):
        connection = get_connection()
        cursor = connection.cursor()

        rows = cursor.execute("""
            SELECT wo.order_id, c.full_name AS client_name,
                   v.make || ' ' || v.model || ' / ' || v.license_plate AS
vehicle_name,
```

```
                e.full_name AS employee_name,
                wo.opened_at, wo.complaint, wo.status, wo.total_amount
            FROM work_orders wo
            JOIN clients c ON c.client_id = wo.client_id
            JOIN vehicles v ON v.vehicle_id = wo.vehicle_id
            LEFT JOIN employees e ON e.employee_id = wo.employee_id
            ORDER BY wo.order_id DESC
        """).fetchall()

        connection.close()

        self.orders_table.setRowCount(len(rows))

        for row_index, row in enumerate(rows):
            values = [
                row["order_id"],
                row["client_name"],
                row["vehicle_name"],
                row["employee_name"] or "-",
                row["opened_at"],
                row["complaint"],
                row["status"],
                f"{row['total_amount']:.2f}"
            ]

            for column_index, value in enumerate(values):
                self.orders_table.setItem(row_index, column_index,
QTableWidgetItem(str(value)))

    def refresh_parts(self):
        connection = get_connection()
        cursor = connection.cursor()

        rows = cursor.execute("""
            SELECT part_name, manufacturer, unit_price, stock_qty, min_qty
            FROM spare_parts
            ORDER BY part_name
        """).fetchall()

        connection.close()

        self.parts_table.setRowCount(len(rows))

        for row_index, row in enumerate(rows):
            values = [
                row["part_name"],
                row["manufacturer"] or "-",
                f"{row['unit_price']:.2f}",
                row["stock_qty"],
                row["min_qty"]
            ]

            for column_index, value in enumerate(values):
                self.parts_table.setItem(row_index, column_index,
QTableWidgetItem(str(value)))

    def refresh_finance(self):
        connection = get_connection()
        cursor = connection.cursor()

        rows = cursor.execute("""
            SELECT i.invoice_id, i.order_id, i.invoice_date,
                i.total_amount, i.payment_status,
```

```
                    COALESCE(SUM(p.amount), 0) AS paid_amount
                FROM invoices i
                LEFT JOIN payments p ON p.invoice_id = i.invoice_id
                GROUP BY i.invoice_id
                ORDER BY i.invoice_id DESC
            """).fetchall()

            connection.close()

            self.finance_table.setRowCount(len(rows))

            for row_index, row in enumerate(rows):
                values = [
                    row["invoice_id"],
                    row["order_id"],
                    row["invoice_date"],
                    f"{row['total_amount']:.2f}",
                    row["payment_status"],
                    f"{row['paid_amount']:.2f}"
                ]

                for column_index, value in enumerate(values):
                    self.finance_table.setItem(row_index, column_index,
QTableWidgetItem(str(value)))

        def refresh_combos(self):
            connection = get_connection()
            cursor = connection.cursor()

            clients = cursor.execute("""
                SELECT client_id, full_name
                FROM clients
                ORDER BY full_name
            """).fetchall()

            vehicles = cursor.execute("""
                SELECT vehicle_id, make, model, license_plate
                FROM vehicles
                ORDER BY license_plate
            """).fetchall()

            employees = cursor.execute("""
                SELECT employee_id, full_name
                FROM employees
                WHERE is_active = 1
                ORDER BY full_name
            """).fetchall()

            connection.close()

            self.client_combo.clear()
            self.vehicle_combo.clear()
            self.employee_combo.clear()

            for client in clients:
                self.client_combo.addItem(client["full_name"], client["client_id"])

            for vehicle in vehicles:
                text = f"{vehicle['make']} {vehicle['model']} /
{vehicle['license_plate']}"
                self.vehicle_combo.addItem(text, vehicle["vehicle_id"])

            for employee in employees:
```

```
            self.employee_combo.addItem(employee["full_name"],
employee["employee_id"])


def apply_styles(app):
    app.setStyleSheet("""
        QWidget {
            background-color: #f5f7fb;
            color: #1f2937;
            font-size: 14px;
            font-family: Arial;
        }

        QLineEdit, QTextEdit, QComboBox {
            background-color: #ffffff;
            border: 1px solid #cbd5e1;
            border-radius: 6px;
            padding: 8px;
        }

        QPushButton {
            background-color: #2563eb;
            color: white;
            border: none;
            border-radius: 6px;
            padding: 9px 14px;
        }

        QPushButton:hover {
            background-color: #1d4ed8;
        }

        QTableWidget {
            background-color: white;
            border: 1px solid #d1d5db;
            gridline-color: #e5e7eb;
        }

        QHeaderView::section {
            background-color: #e5edf7;
            border: 1px solid #d1d5db;
            padding: 7px;
            font-weight: 600;
        }

        QLabel#title {
            font-size: 18px;
            font-weight: 600;
            color: #1e3a8a;
        }

        QLabel#hint {
            color: #6b7280;
            font-size: 12px;
        }

        QLabel#sectionTitle {
            font-size: 17px;
            font-weight: 600;
            color: #111827;
            padding: 8px;
        }
```

```
        QLabel#card {
            background-color: white;
            border: 1px solid #d1d5db;
            border-radius: 10px;
            padding: 20px;
            font-size: 16px;
        }
    """)


def main():
    init_database()
    seed_database()

    app = QApplication(sys.argv)
    apply_styles(app)

    window = LoginWindow()
    window.show()

    sys.exit(app.exec())


if __name__ == "__main__":
    main()
```

ДОДАТОК Г

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Піддубний Владислав Олегович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Програмне забезпечення облікової системи для станції технічного обслуговування» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [+] інструменти ШІ (вказати назву: **ChatGPT**, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
        - [ ] перекладу іншомовних джерел;
        - [+] стилістичного редагування та перевірки граматики;
        - [ ] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: __пошуку літератури, допомоги з ідеями_____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____ **Владислав Піддубний**
(підпис)