Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 - «Інженерія програмного забезпечення»

На тему: Розробка веб-орієнтованої інформаційної системи обліку замовлень

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-23ск
_____ / Д.О. Ємельянович /

Керівник кваліфікаційної роботи _____ / А.М. Гриценко /

Завідувач кафедри _____ / А.М. Стрюк /

Кривий Ріг
2026

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Освітньо-кваліфікаційний рівень: бакалавр

Спеціальність: 121 - Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А.М. Стрюк

«___»_____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-23ск Ємельяновичу Денису Олександровичу

1. Тема: Розробка веб-орієнтованої інформаційної системи обліку замовлень затверджено наказом по КНУ №283 від «28» травня 2026 р.

2. Термін подання студентом закінченого проекту «08» червня 2026 р.

3. Вихідні дані по роботі: *Пояснювальна записка: 117 сторінки, 25 рисунків, 1 додаток, 24 використаних у роботі джерел.*

4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): Аналіз існуючих програмних рішень в галузі обліку замовлень. Проектування та розробка інформаційно системи обробки замовлень. Тестування системи веб-орієнтованої системи обліку замовлень.

5. Перелік графічного демонстраційного матеріалу: Функціональна схема процесу обліку замовлень у ремонтній майстерні. Схема взаємодії ролей у процесі обліку замовлень. Схема функціональних модулів інформаційної системи обліку замовлень. Функціональна схема веб-орієнтованої інформаційної системи обліку замовлень. Структцура бази даних інформаційної системи. Алгоритми створення та обробки замовлення.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Формулювання мети та задач роботи	18.02.2026-20.02.2026
2	Аналіз інформаційних джерел	21.02.2026-09.03.2026
3	Визначення вимог до програмного забезпечення	10.03.2026-18.03.2026
4	Розробка функціональної схеми	19.03.2026-23.03.2026
5	Розробка алгоритмів	24.03.2026-31.03.2026
6	Розробка бази даних	01.04.2026-10.04.2026
7	Розробка програмного забезпечення	10.04.2026-13.05.2026
8	Тестування програмного забезпечення	14.05.2026-20.05.2026
9	Оформлення пояснювальної записки	21.05.2026-31.06.2026
10	Розробка демонстраційних матеріалів	01.06.2026-10.06.2026

Дата видачі завдання: « 18 » лютого 2026 р.

Студент _____ / Д.О. Ємельянович /

Керівник роботи _____ / А.М. Гриценко /

РЕФЕРАТ

ВЕБ-ОРІЄНТОВАНА ІНФОРМАЦІЙНА СИСТЕМА, ОБЛІК ЗАМОВЛЕНЬ, РЕМОНТНА МАЙСТЕРНЯ, REACT, PYTHON, FASTAPI, MYSQL, REST API, БАЗА ДАНИХ, РОЛЬОВИЙ ДОСТУП.

Пояснювальна записка: 117 с., 31 табл., 26 рис., 1 дод., 24 джерела.

Об’єктом розробки є процеси обліку та супроводження замовлень у ремонтній майстерні. Метою роботи є створення веб-орієнтованої інформаційної системи для обліку клієнтів, пристроїв, заявок, замовлень, результатів діагностики, ремонтних робіт, запчастин, статусів і оплат.

У роботі проаналізовано предметну область та існуючі програмні рішення, сформовано функціональні вимоги, розроблено архітектуру, алгоритми, UML-діаграми та структуру бази даних. Клієнтську частину реалізовано засобами React, серверну — мовою Python із використанням FastAPI, а збереження даних забезпечує MySQL. Обмін даними між компонентами виконується через REST API.

Система підтримує подання заявки, створення й редагування замовлень, призначення майстра, внесення результатів діагностики, облік робіт і запчастин, зміну статусів, перегляд стану замовлення клієнтом та формування зведеної інформації для керівника. Для внутрішніх користувачів передбачено авторизацію та розмежування доступу відповідно до ролі.

Проведене функціональне та інтеграційне тестування підтвердило працездатність основних функцій, правильність збереження даних і коректність взаємодії між React, FastAPI та MySQL. Практичне значення роботи полягає у скороченні кількості ручних операцій, підвищенні зручності контролю замовлень і централізованому збереженні інформації. Галуззю застосування є ремонтні майстерні та сервісні центри.

ABSTRACT

WEB-ORIENTED INFORMATION SYSTEM, ORDER ACCOUNTING, REPAIR WORKSHOP, REACT, PYTHON, FASTAPI, MYSQL, REST API, DATABASE, ROLE-BASED ACCESS.

Explanatory note: 117 pages, 31 tables, 26 figures, 1 appendices, 24 references.

The object of development is the processes of accounting and supporting orders in a repair workshop. The purpose of the work is to create a web-oriented information system for accounting clients, devices, requests, orders, diagnostic results, repair works, spare parts, statuses and payments.

The subject area and existing software solutions were analysed, functional requirements were defined, and the architecture, algorithms, UML diagrams and database structure were developed. The client side was implemented using React, the server side was developed in Python using FastAPI, and MySQL was used for data storage. Data exchange between the components is performed through a REST API.

The system supports submitting requests, creating and editing orders, assigning a technician, entering diagnostic results, accounting for repair works and spare parts, changing statuses, viewing the order status by the client and generating summary information for the manager. Authentication and role-based access control are provided for internal users.

Functional and integration testing confirmed the operability of the main functions, the correctness of data storage and the proper interaction between React, FastAPI and MySQL. The practical value of the work lies in reducing the number of manual operations, improving order control and providing centralised information storage. The application area includes repair workshops and service centres.

ЗМІСТ

ВСТУП

У сучасних умовах діяльність ремонтних майстерень пов’язана з необхідністю швидкої обробки значної кількості замовлень, контролю стану їх виконання, обліку клієнтів, виконавців, використаних матеріалів і вартості наданих послуг. Для невеликих та середніх майстерень часто характерним є ведення обліку за допомогою паперових журналів, електронних таблиць або неструктурованих повідомлень у месенджерах. Такий підхід ускладнює пошук потрібної інформації, підвищує ризик втрати даних, створює незручності під час контролю термінів виконання робіт та не забезпечує достатньої прозорості процесу обслуговування клієнтів.

Особливістю ремонтної майстерні як предметної області є те, що кожне замовлення має не лише загальні дані про клієнта, але й інформацію про об’єкт ремонту, опис несправності, попередню діагностику, статус виконання, використані комплектуючі, вартість робіт і дату видачі готового замовлення. На відміну від звичайної системи продажу товарів, у майстерні важливим є поетапне супроводження замовлення: від моменту приймання пристрою до завершення ремонту та передачі його клієнту. Саме тому система обліку замовлень для майстерні повинна враховувати специфіку ремонтного процесу, а не лише забезпечувати стандартне створення та збереження записів.

Актуальність теми кваліфікаційної роботи полягає в необхідності автоматизації процесу обліку замовлень у ремонтній майстерні шляхом створення веб-орієнтованої інформаційної системи. Використання такої системи дозволяє зменшити кількість ручних операцій, прискорити обробку замовлень, підвищити точність збереження даних, забезпечити швидкий доступ до інформації про клієнтів, замовлення та стан виконання ремонтних робіт. Веб-орієнтований формат системи є зручним, оскільки користувачі можуть працювати з нею через браузер без необхідності встановлення додаткового програмного забезпечення на робочі комп’ютери.

Метою кваліфікаційної роботи є розробка веб-орієнтованої інформаційної системи обліку замовлень ремонтної майстерні, яка забезпечує автоматизацію основних процесів приймання, супроводження, редагування та контролю виконання замовлень.

Для досягнення поставленої мети необхідно виконати такі завдання:

– проаналізувати предметну область ремонтної майстерні та визначити особливості процесу обліку замовлень;

– виконати огляд існуючих програмних рішень для обліку замовлень і визначити їх основні переваги та недоліки;

– сформулювати функціональні та нефункціональні вимоги до веб-орієнтованої інформаційної системи;

– розробити функціональну схему системи та алгоритм її роботи;

– спроєктувати архітектуру програмного забезпечення з урахуванням клієнтської, серверної частини та бази даних;

– розробити структуру бази даних для збереження інформації про користувачів, клієнтів, замовлення, ремонтні роботи, статуси та комплектуючі;

– реалізувати клієнтську частину системи з використанням бібліотеки React [1];

– реалізувати серверну частину системи засобами мови програмування Python [2];

– реалізувати базу даних з використанням MySQL [3];

– виконати тестування основних функціональних можливостей системи;

– підготувати інструкцію користувача та екранні форми, що демонструють роботу розробленого програмного забезпечення.

Об’єктом розробки є процес обліку та супроводження замовлень у ремонтній майстерні.

Предметом розробки є веб-орієнтована інформаційна система для автоматизації обліку замовлень, клієнтів, ремонтних робіт, статусів виконання та вартості послуг.

У процесі виконання роботи використовуються методи аналізу предметної області, проєктування програмного забезпечення, моделювання структури бази даних, розробки клієнт-серверних веб-застосунків та тестування програмних систем. Для реалізації програмного забезпечення обрано React для створення клієнтського інтерфейсу [1], Python для розробки серверної логіки [2] та MySQL для організації збереження даних [3].

Практичне значення роботи полягає у створенні програмного засобу, який може бути використаний у роботі ремонтної майстерні для зручного ведення обліку замовлень, контролю етапів виконання ремонту, збереження інформації про клієнтів та формування історії виконаних робіт. Розроблена система може бути адаптована для майстерень різного типу, зокрема майстерень з ремонту побутової техніки, комп’ютерної техніки, мобільних пристроїв або іншого обладнання.

Результатом виконання кваліфікаційної роботи є веб-орієнтована інформаційна система обліку замовлень ремонтної майстерні, що забезпечує створення, перегляд, редагування, пошук і контроль замовлень, а також збереження необхідної інформації в базі даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз діяльності ремонтної майстерні

Ремонтна майстерня є підприємством або структурним підрозділом, діяльність якого спрямована на приймання, діагностику, ремонт, обслуговування та видачу клієнтам відремонтованих пристроїв. Залежно від спеціалізації майстерня може виконувати ремонт побутової техніки, комп’ютерної техніки, мобільних пристроїв, електроінструментів, офісного обладнання або інших технічних засобів. Незважаючи на різні напрями роботи, більшість ремонтних майстерень мають подібну організацію основних бізнес-процесів.

Основним процесом діяльності ремонтної майстерні є обробка замовлення клієнта. Замовлення формується після звернення клієнта до майстерні та містить інформацію про клієнта, об’єкт ремонту, опис несправності, дату приймання, попередній стан пристрою, орієнтовну вартість робіт, відповідального виконавця та поточний статус виконання. У процесі виконання ремонту замовлення може змінювати свій стан: від нового або прийнятого до виконаного та виданого клієнту. Таким чином, замовлення в ремонтній майстерні є не просто записом про надання послуги, а основною одиницею обліку всього виробничого процесу.

Типовий процес роботи ремонтної майстерні починається з приймання звернення від клієнта. На цьому етапі працівник майстерні фіксує контактні дані клієнта, інформацію про пристрій, опис заявленої несправності та зовнішній стан об’єкта ремонту. Після цього замовленню присвоюється статус, наприклад «Нове» або «Прийнято». Далі пристрій передається майстру для проведення діагностики. За результатами діагностики визначається причина несправності, перелік необхідних робіт, потреба в комплектуючих і попередня або остаточна вартість ремонту.

Після узгодження умов ремонту із клієнтом замовлення переходить до етапу виконання робіт. На цьому етапі важливо мати можливість відстежувати, хто саме виконує ремонт, які дії вже виконано, які комплектуючі використано та на якому етапі перебуває замовлення. У разі відсутності потрібних деталей замовлення може отримати статус «Очікує комплектуючі». Після завершення ремонту працівник майстерні фіксує виконані роботи, загальну вартість, дату завершення та переводить замовлення у статус «Виконано». Остаточним етапом є видача пристрою клієнту та закриття замовлення.

Для ремонтної майстерні характерна необхідність зберігати історію виконаних замовлень. Це дозволяє швидко переглядати попередні звернення клієнта, визначати, які роботи виконувалися раніше, які комплектуючі використовувалися та чи звертався клієнт повторно з тією самою несправністю. Наявність такої історії підвищує якість обслуговування та спрощує роботу працівників майстерні.

У практичній діяльності невеликих майстерень облік часто ведеться вручну: у паперових журналах, таблицях Microsoft Excel або за допомогою повідомлень у месенджерах. Microsoft Excel є поширеним інструментом для роботи з електронними таблицями та аналізу даних [4], однак такий спосіб організації роботи має низку недоліків. Зокрема, ускладнюється пошук потрібного замовлення, виникає ризик помилок під час введення даних, складно контролювати поточні статуси ремонту, немає зручного розподілу замовлень між майстрами, а також відсутня централізована база клієнтів і виконаних робіт. Крім того, ручний облік не завжди дозволяє швидко отримати узагальнену інформацію про кількість активних, завершених або прострочених замовлень.

Особливістю діяльності ремонтної майстерні є залежність якості обслуговування не лише від професійності майстрів, а й від правильної організації інформаційних процесів. Якщо дані про замовлення зберігаються неструктуровано, працівникам складніше контролювати терміни виконання,

уточнювати деталі ремонту та надавати клієнту актуальну інформацію. Це може призводити до затримок, втрати інформації, дублювання записів і зниження рівня довіри клієнтів.

Для ефективної роботи ремонтної майстерні доцільно використовувати інформаційну систему, яка забезпечує централізоване збереження даних, зручний доступ до замовлень, фіксацію змін статусів, облік клієнтів, виконавців, ремонтних робіт і використаних комплектуючих. Спеціалізовані системи для сервісного бізнесу передбачають роботу із замовленнями, клієнтами, запасами, графіком виконання робіт, рахунками та платежами [5]. Така система повинна підтримувати основні операції зі створення, перегляду, редагування, пошуку та закриття замовлень. Окрему увагу слід приділити простоті інтерфейсу, оскільки користувачами системи можуть бути адміністратори, менеджери або майстри, для яких важлива швидкість і зручність роботи.

Отже, аналіз діяльності ремонтної майстерні показує, що основною інформаційною одиницею її роботи є замовлення на ремонт. Його облік охоплює дані про клієнта, об’єкт ремонту, несправність, етапи виконання, виконавця, комплектуючі та вартість послуг. Наявність веб-орієнтованої інформаційної системи дозволить упорядкувати ці дані, зменшити кількість ручних операцій і підвищити ефективність організації роботи майстерні.

1.2 Аналіз процесу обліку замовлень

Процес обліку замовлень є центральним бізнес-процесом ремонтної майстерні, оскільки саме навколо замовлення формується основна інформація про клієнта, об’єкт ремонту, характер несправності, виконавця, використані матеріали, поточний стан роботи та кінцеву вартість послуги. Подібний підхід використовується у спеціалізованих програмних рішеннях для сервісного бізнесу, де замовлення виступає основним елементом обліку робіт, клієнтів, статусів і платежів [5]. Якість організації цього процесу безпосередньо

впливає на швидкість обслуговування клієнтів, контроль завантаженості майстрів, своєчасність виконання робіт і збереження історії звернень.

У загальному вигляді процес обліку замовлення в ремонтній майстерні починається з надходження звернення від клієнта. Клієнт повідомляє інформацію про пристрій, який потребує ремонту, а також описує виявлену несправність. Працівник майстерні створює нове замовлення, вносить контактні дані клієнта, опис об’єкта ремонту, дату приймання, попередній опис проблеми та встановлює початковий статус замовлення. Після цього замовлення передається на діагностику або одразу призначається відповідальному майстру.

На рисунку 1.1 зображено загальну функціональну схему процесу обліку замовлень у ремонтній майстерні.

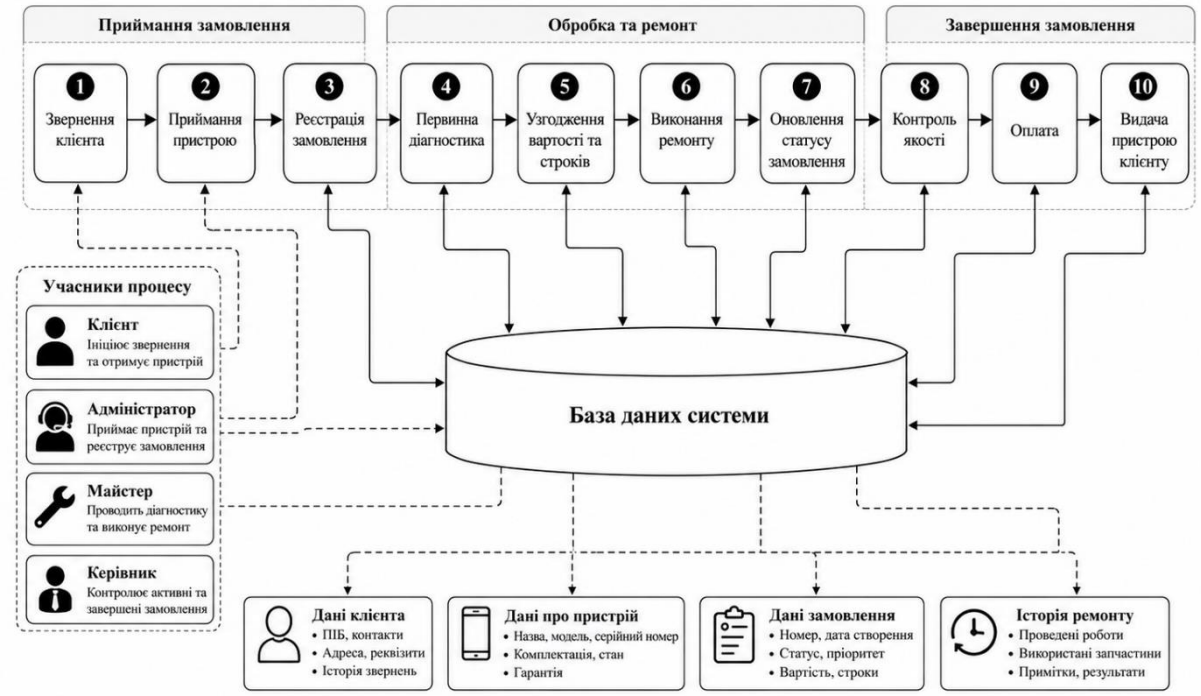

Рисунок 1.1 – Функціональна схема процесу обліку замовлень у ремонтній майстерні

Відповідно до наведеної схеми, процес обліку замовлення можна поділити на декілька основних етапів: приймання замовлення, діагностика, узгодження вартості та строків, виконання ремонтних робіт, контроль якості,

оплата та видача пристрою клієнту. На кожному з цих етапів у системі повинна зберігатися актуальна інформація про стан замовлення.

Першим етапом є приймання замовлення. На цьому етапі адміністратор або інший відповідальний працівник створює запис про нове замовлення. До такого запису вносяться прізвище та ім’я клієнта, номер телефону, тип пристрою, модель, серійний номер за наявності, опис несправності, зовнішній стан пристрою, дата приймання та коментарі. Після створення замовлення йому присвоюється унікальний номер, за яким його можна швидко знайти в системі.

Другим етапом є діагностика. На цьому етапі майстер перевіряє об’єкт ремонту, визначає причину несправності та вносить у систему результати діагностики. Якщо ремонт можливий, формується перелік необхідних робіт, орієнтовна вартість і потреба у комплектуючих. Якщо потрібні деталі відсутні, замовлення може отримати статус «Очікує комплектуючі». Якщо ремонт недоцільний або неможливий, у системі фіксується відповідний коментар.

Третім етапом є узгодження ремонту з клієнтом. Для ремонтної майстерні цей етап є важливим, оскільки фактична вартість ремонту часто стає відомою лише після діагностики. Працівник майстерні повідомляє клієнту вартість, строки та умови виконання робіт. Після погодження замовлення переходить до етапу виконання ремонту.

Четвертим етапом є безпосереднє виконання ремонтних робіт. Майстер виконує ремонт, за потреби зазначає використані комплектуючі, додає коментарі щодо виконаних дій і змінює статус замовлення. У спеціалізованих системах для сервісного бізнесу статуси використовуються для контролю етапів виконання робіт і відстеження стану замовлень [5]. Наприклад, замовлення може мати такі статуси: «Нове», «Прийнято», «На діагностиці», «В роботі», «Очікує комплектуючі», «Виконано», «Видано клієнту», «Скасовано». Наявність статусів дозволяє швидко визначити, на якому етапі перебуває конкретне замовлення.

Після завершення ремонту здійснюється перевірка результату роботи. Якщо ремонт виконано успішно, замовлення переводиться у статус «Виконано». Далі клієнт здійснює оплату, отримує відремонтований пристрій, а замовлення закривається зі статусом «Видано клієнту». У системі при цьому зберігається повна історія виконаних робіт, що дає змогу в майбутньому переглянути інформацію про попередні звернення клієнта.

Окрему увагу в процесі обліку замовлень необхідно приділити взаємодії різних ролей користувачів. Для інформаційної системи ремонтної майстерні доцільно передбачити такі основні ролі: адміністратор, керівник, майстер і клієнт. Кожна з цих ролей має власні функції та рівень доступу до даних.

Адміністратор відповідає за створення та редагування замовлень, внесення інформації про клієнтів, призначення майстра, зміну статусів і пошук необхідних записів. Керівник контролює загальну роботу майстерні, переглядає всі замовлення, аналізує завантаженість майстрів, контролює кількість виконаних і активних замовлень. Майстер працює із замовленнями, які були йому призначені, вносить результати діагностики, опис виконаних робіт і змінює технічний статус ремонту. Клієнт не обов’язково повинен мати повний доступ до системи, але для нього може бути передбачена можливість отримання інформації про стан замовлення за його номером або через звернення до адміністратора.

На рисунку 1.2 зображено схему взаємодії основних ролей у процесі обліку замовлень ремонтної майстерні.

Зі схеми видно, що адміністратор є основною роллю, яка забезпечує первинне внесення даних до системи. Він створює замовлення, вказує інформацію про клієнта та передає замовлення майстру. Майстер, у свою чергу, працює з технічною частиною замовлення: проводить діагностику, зазначає виконані ремонтні роботи та оновлює статус. Керівник має доступ до зведеної інформації та може контролювати виконання робіт без необхідності самостійно редагувати кожне замовлення. Клієнт взаємодіє із системою опосередковано або через окремий інтерфейс перегляду статусу замовлення.

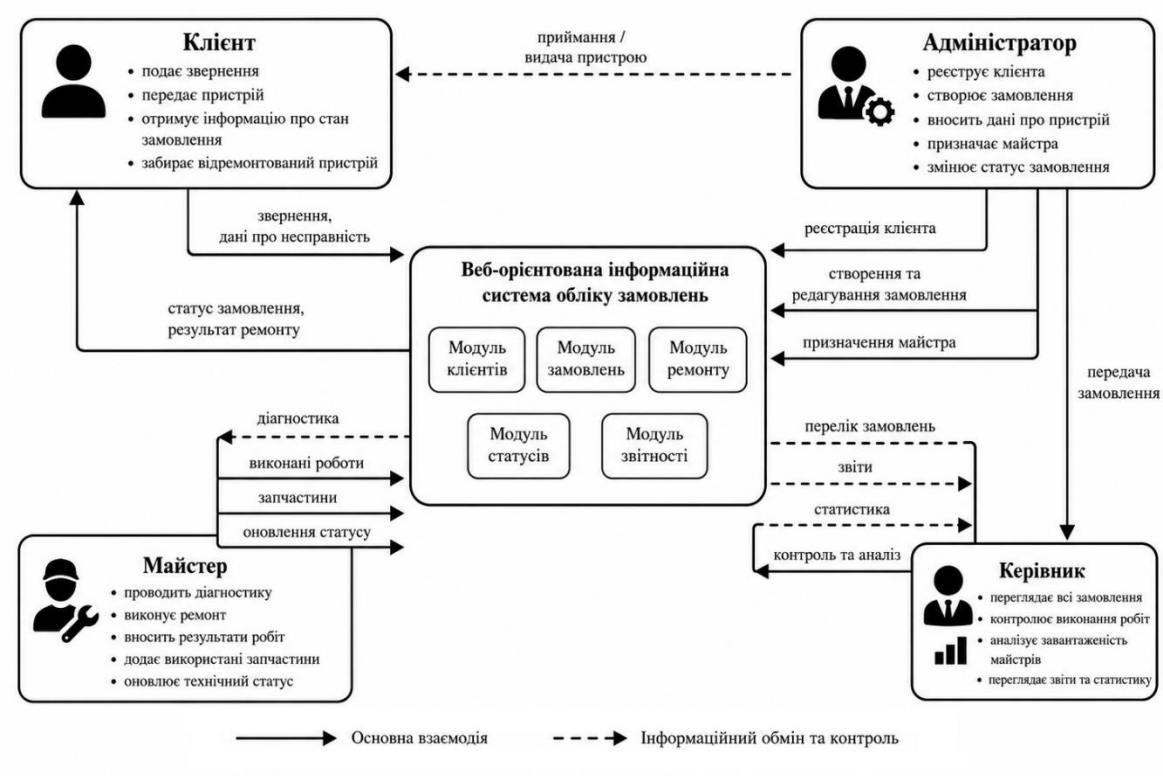

Рисунок 1.2 – Схема взаємодії ролей у процесі обліку замовлень

Для більш детального аналізу доцільно розглянути інформаційні потоки, які виникають під час обліку замовлення. Основними вхідними даними є дані клієнта, опис несправності, інформація про пристрій, результати діагностики, перелік робіт, використані комплектуючі та оплата. Вихідними даними є картка замовлення, поточний статус ремонту, історія виконаних робіт, перелік активних замовлень, звітна інформація для керівника та повідомлення для клієнта. Такий підхід відповідає логіці спеціалізованих систем для сервісного бізнесу, у яких облік замовлень пов’язаний із клієнтами, виконавцями, статусами, роботами та платежами [5].

На рисунку 1.3 наведено інформаційну модель взаємодії між користувачами та системою обліку замовлень.

Наведена схема показує, що веб-орієнтована інформаційна система виступає центральним елементом обміну даними між усіма учасниками процесу. Адміністратор вносить і редагує дані, майстер доповнює замовлення технічною інформацією, керівник переглядає зведені показники, а клієнт

отримує інформацію про стан виконання замовлення. Така організація дозволяє уникнути дублювання інформації та зменшує ризик втрати важливих даних.

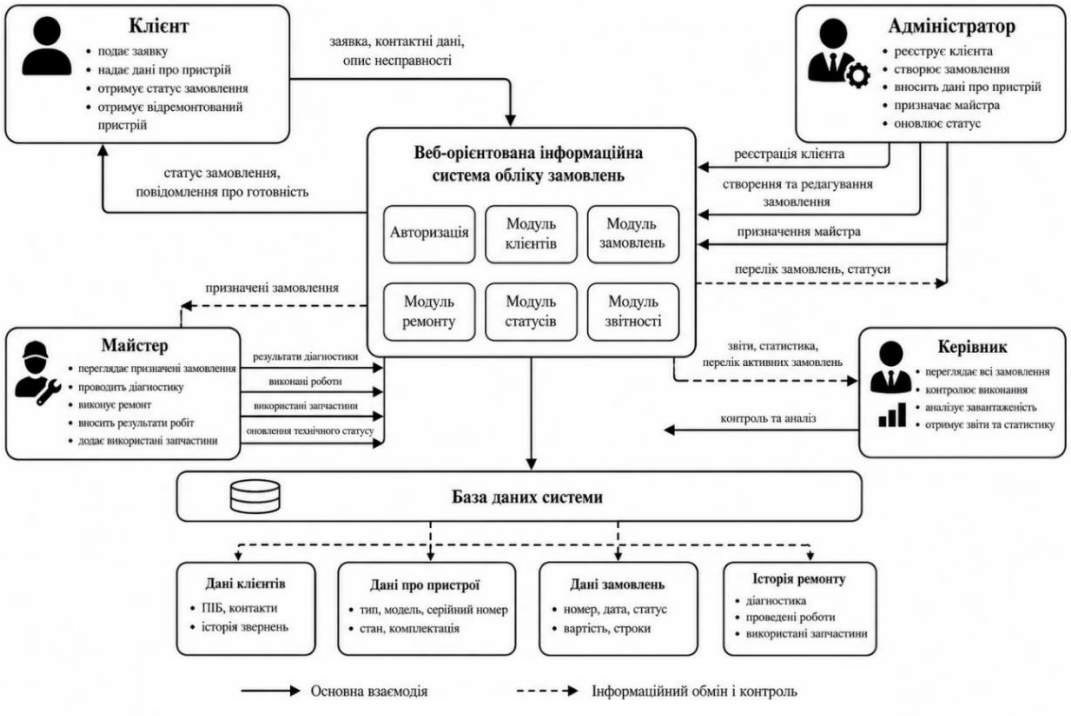

Рисунок 1.3 – Інформаційна схема взаємодії користувачів із системою обліку замовлень

У процесі обліку замовлень важливо забезпечити не лише збереження даних, а й логічний контроль переходів між статусами. Наприклад, замовлення не може бути видане клієнту до завершення ремонту, а ремонт не може бути розпочатий без попереднього приймання замовлення. Тому в системі доцільно передбачити послідовність зміни статусів, яка відповідає реальному процесу роботи майстерні.

На рисунку 1.4 зображено приклад життєвого циклу замовлення в ремонтній майстерні.

Життєвий цикл замовлення відображає послідовність його переходу між основними станами. Спочатку замовлення створюється в системі та отримує статус «Нове». Після приймання пристрою воно переходить у статус

«Прийнято». Далі замовлення передається на діагностику, після чого може перейти в статус «В роботі», «Очікує комплектуючі» або «Скасовано». Після завершення ремонту замовлення отримує статус «Виконано», а після передачі пристрою клієнту — «Видано клієнту». Використання статусів замовлення є типовим підходом для систем керування сервісними процесами та дозволяє контролювати етапи виконання робіт [5].

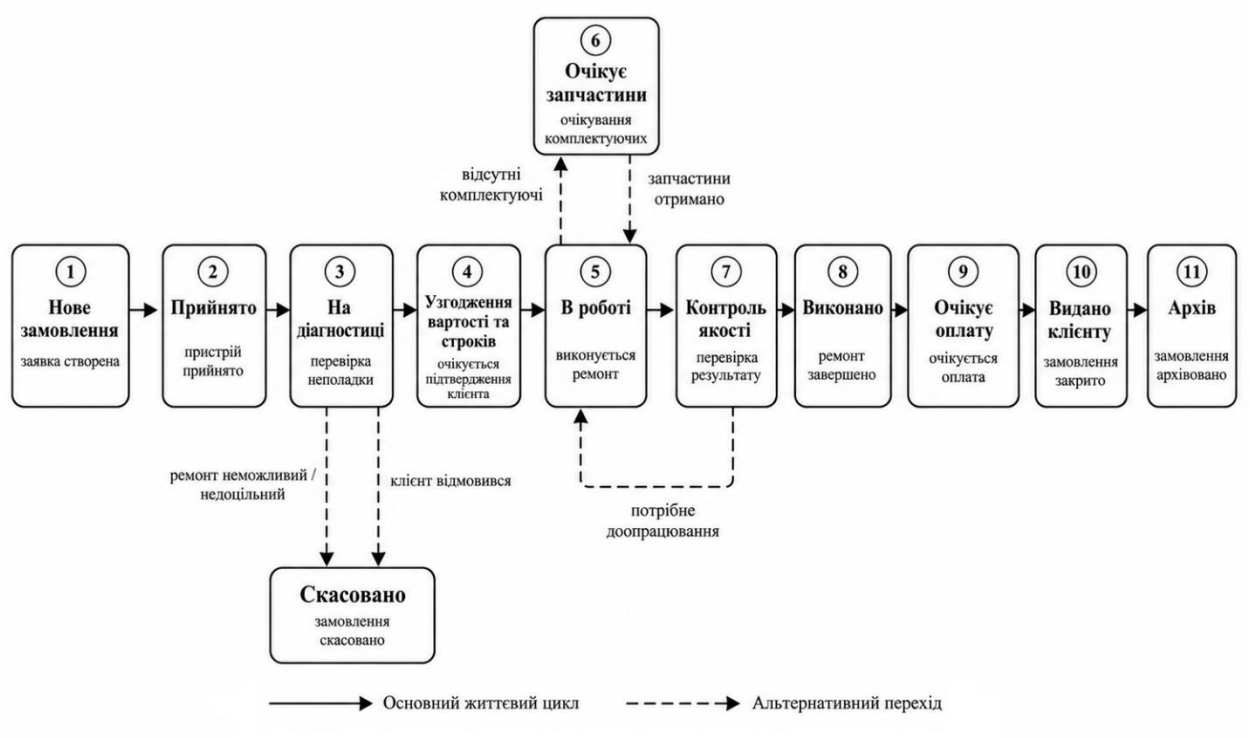

Рисунок 1.4 – Життєвий цикл замовлення в ремонтній майстерні

Під час аналізу процесу обліку замовлень було визначено, що для ефективної роботи ремонтної майстерні система повинна забезпечувати такі можливості:

– створення нового замовлення з фіксацією даних клієнта та об’єкта ремонту;

– перегляд списку активних і завершених замовлень;

– пошук замовлення за номером, клієнтом, телефоном, датою або статусом;

– редагування інформації про замовлення;

– призначення відповідального майстра;

– зміна статусу замовлення відповідно до етапу виконання робіт;

– фіксація результатів діагностики та виконаних ремонтних робіт;

– облік використаних комплектуючих;

– розрахунок вартості ремонту;

– перегляд історії звернень клієнта;

– формування зведеної інформації для керівника.

Таким чином, аналіз процесу обліку замовлень показує, що ремонтна майстерня потребує централізованої інформаційної системи, яка дозволить упорядкувати роботу з клієнтами, замовленнями, майстрами та статусами виконання робіт. Основною перевагою такого підходу є зменшення ручного введення та дублювання інформації, підвищення швидкості пошуку даних, покращення контролю за виконанням замовлень і забезпечення збереження історії ремонтних робіт.

1.3 Огляд існуючих програмних рішень для обліку замовлень

Для обґрунтування необхідності розробки веб-орієнтованої інформаційної системи обліку замовлень ремонтної майстерні доцільно розглянути існуючі програмні рішення, які використовуються для автоматизації обліку клієнтів, замовлень, ремонтних робіт, складських залишків, оплат і звітності. Аналіз таких систем дозволяє визначити типові функціональні можливості, переваги та недоліки готових рішень, а також сформувати вимоги до власної програмної розробки.

Під час аналізу були розглянуті такі програмні рішення:

– RO App / RemOnline [6];

– Orderry [5];

– Odoo [7];

– Zoho Inventory [8].

Зазначені системи мають різний рівень спеціалізації. RO App / RemOnline та Orderry орієнтовані переважно на сервісні компанії, ремонтні майстерні та організації, які працюють із замовленнями на обслуговування або ремонт [5; 6]. Odoo та Zoho Inventory є більш універсальними системами, які можуть застосовуватися для обліку замовлень, товарів, клієнтів і складських операцій, однак потребують додаткового налаштування під специфіку ремонтної майстерні [7; 8].

RO App / RemOnline є веб-орієнтованою CRM-системою, призначеною для сервісних центрів, ремонтних майстерень та інших підприємств, що працюють із замовленнями клієнтів. Система дозволяє вести облік клієнтів, створювати замовлення, контролювати етапи виконання робіт, працювати зі складом, оплатами та звітністю [6].

Інтерфейс системи орієнтований на роботу з переліком замовлень. Користувач може переглядати активні та завершені замовлення, фільтрувати їх за статусами, відповідальними працівниками, датами або іншими параметрами. Такий підхід є зручним для сервісної діяльності, оскільки працівники майстерні можуть швидко визначити, які замовлення перебувають у роботі, які очікують комплектуючих, а які вже готові до видачі клієнту.

На рисунку 1.5 наведено приклад інтерфейсу системи RO App / RemOnline із переліком замовлень.

До переваг RO App / RemOnline можна віднести спеціалізацію на сервісних процесах, наявність обліку клієнтів, замовлень, складу, оплат і звітів [6]. Система добре підходить для реальних майстерень, оскільки враховує процес роботи із замовленням від його створення до завершення. Недоліком такого рішення для невеликої майстерні може бути надмірна кількість функцій, залежність від готового хмарного сервісу, платна модель використання та обмежені можливості зміни внутрішньої логіки під індивідуальні потреби конкретної організації.

Orderry є програмною платформою для управління сервісним бізнесом, зокрема ремонтними майстернями. Система дозволяє працювати із

замовленнями, клієнтами, розкладом, складом, оплатами та історією виконання робіт [5]. Для кожного замовлення може зберігатися інформація про відповідального виконавця, статус, використані запчастини, оплату та історію змін.

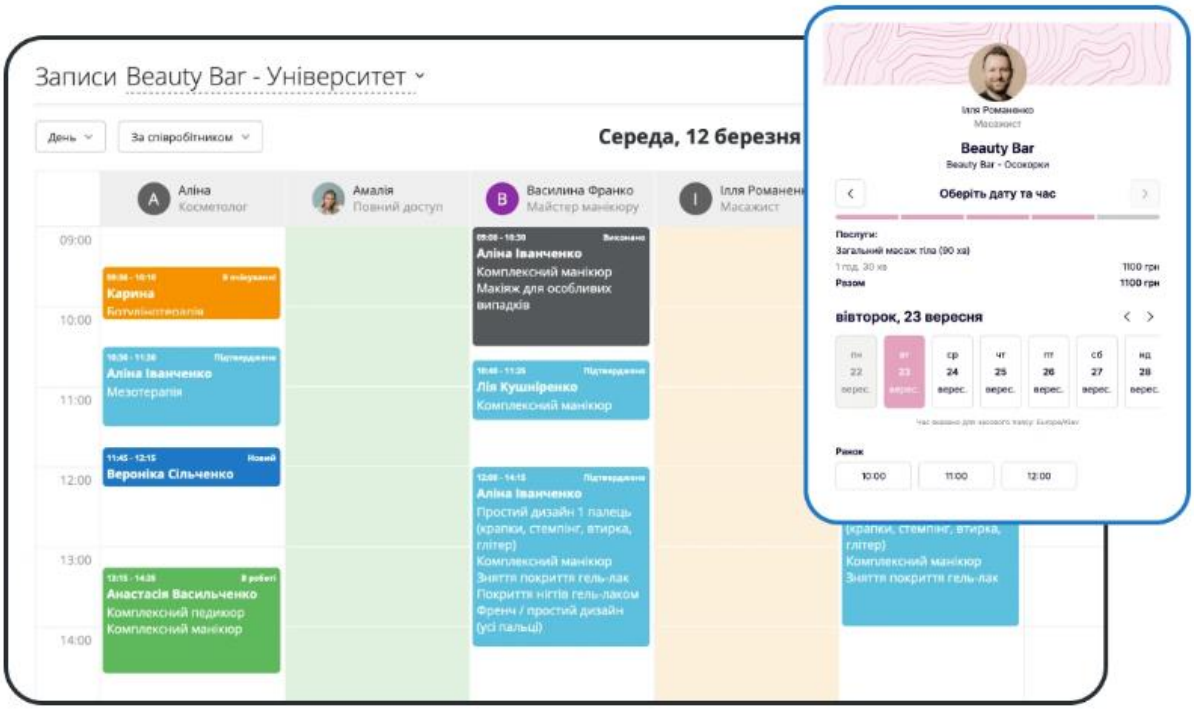

Рисунок 1.5 – Приклад інтерфейсу системи RO App / RemOnline

Особливістю Orderry є орієнтація на повний цикл роботи із замовленням. Замовлення супроводжується від моменту звернення клієнта до завершення ремонту та оплати [5]. Для майстерні це є важливим, оскільки дозволяє контролювати не лише факт створення замовлення, а й усі проміжні етапи його виконання.

На рисунку 1.6 зображено приклад інтерфейсу Orderry, де відображено роботу із замовленнями або карткою ремонтного замовлення.

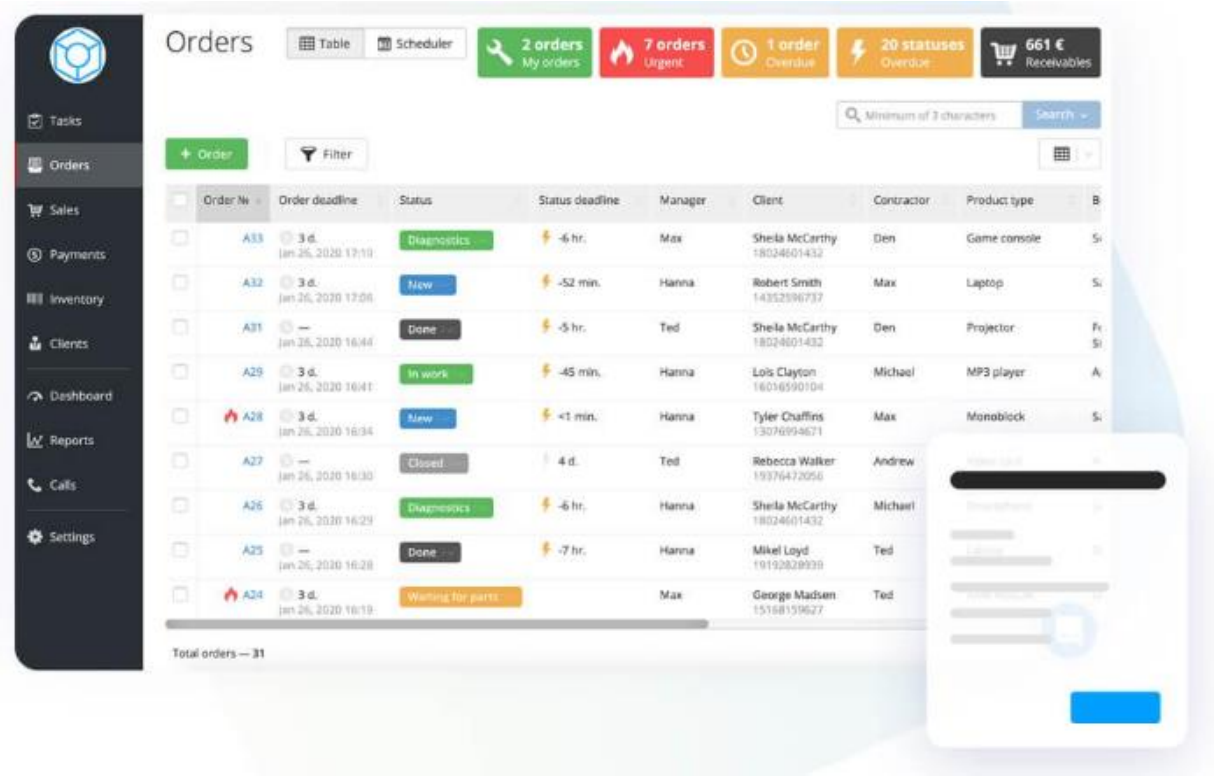

Рисунок 1.6 – Приклад інтерфейсу системи Orderry

Перевагами Orderry є зручна робота з ремонтними замовленнями, підтримка статусів, можливість контролю завантаженості працівників, ведення історії змін і наявність інструментів для роботи зі складом та оплатами [5]. Недоліком є те, що система є готовим комерційним продуктом і може мати функції, які не потрібні для невеликої ремонтної майстерні. Крім того, адаптація інтерфейсу та бізнес-логіки під конкретні навчальні або локальні потреби є обмеженою.

Odoo є комплексною ERP-системою, яка складається з окремих модулів для управління продажами, складом, клієнтами, виробництвом, обслуговуванням, фінансами та іншими процесами [7]. У контексті обліку замовлень ремонтної майстерні Odoo може бути використана для ведення клієнтської бази, контролю замовлень, роботи зі складом і формування звітності.

На відміну від вузькоспеціалізованих рішень для сервісних центрів, Odoo є більш універсальною платформою. Це дозволяє використовувати її в різних галузях, але для ремонтної майстерні така система потребує

додаткового налаштування. Зокрема, необхідно адаптувати структуру замовлення, статуси ремонту, облік запчастин, виконавців і логіку переходу між етапами ремонту.

На рисунку 1.7 зображено приклад інтерфейсу Odoo, що демонструє роботу з модулями замовлень, обслуговування або складського обліку.

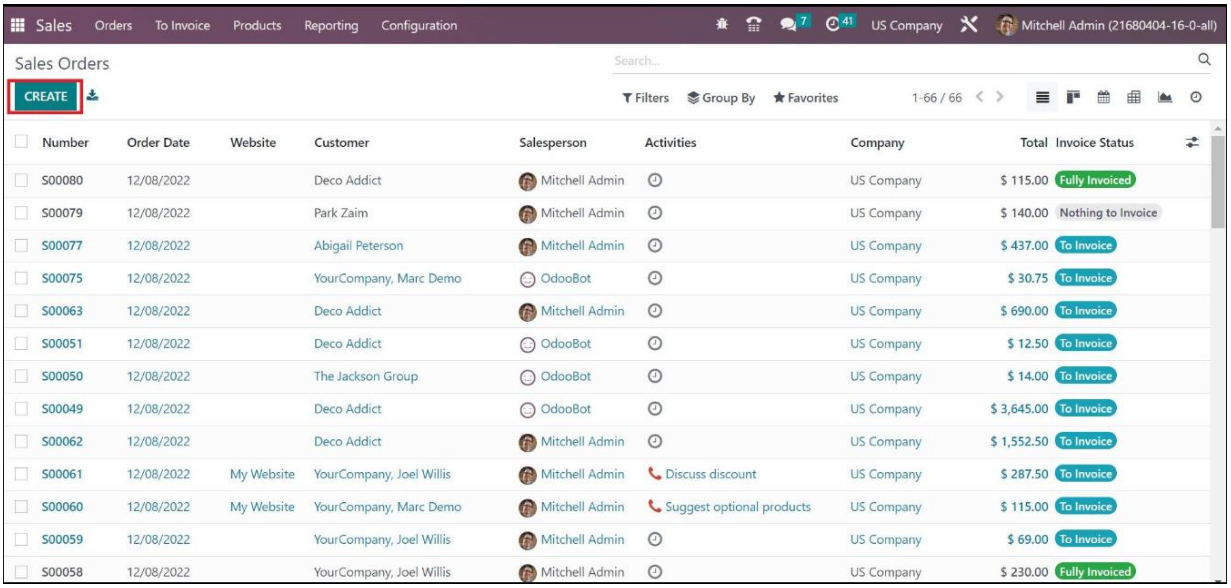

Рисунок 1.7 – Приклад інтерфейсу системи Odoo

Перевагою Odoo є модульність і можливість розширення функціональності. Система може використовуватися не лише для обліку замовлень, а й для комплексного управління підприємством. Недоліком є складність упровадження, необхідність налаштування під конкретну предметну область і надмірність функціоналу для невеликої ремонтної майстерні.

Zoho Inventory є веб-орієнтованою системою для управління замовленнями, складськими залишками, продажами, відправленнями та платежами. Система більше орієнтована на торгівлю та складський облік, однак її функціональні можливості можуть бути частково використані для організації обліку замовлень.

У контексті ремонтної майстерні Zoho Inventory може бути корисною для контролю замовлень, клієнтів, оплат і наявності запчастин. Проте система

не є спеціалізованою саме для ремонтної діяльності. У ній відсутня типова для майстерні логіка діагностики, ремонту, фіксації несправностей, історії ремонтних робіт і передачі замовлення між адміністратором та майстром.

На рисунку 1.8 наведено приклад інтерфейсу Zoho Inventory, який демонструє роботу із замовленнями або складським обліком.

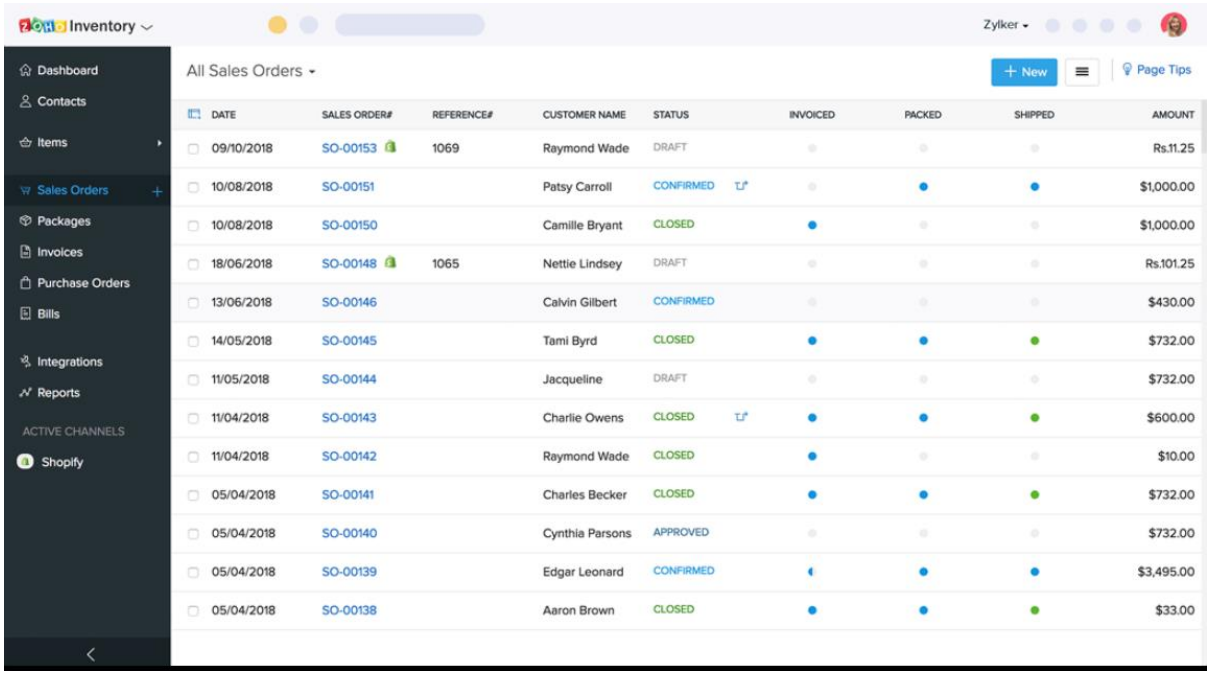

Рисунок 1.8 – Приклад інтерфейсу системи Zoho Inventory

Перевагами Zoho Inventory є зручність роботи із замовленнями, складом і продажами, наявність веб-інтерфейсу та підтримка інтеграцій з іншими сервісами. Недоліком для теми цієї кваліфікаційної роботи є те, що система більше орієнтована на товарні замовлення та логістику, а не на процес ремонту пристроїв.

Порівняльну характеристику розглянутих програмних рішень наведено в таблиці 1.1.

За результатами аналізу можна зробити висновок, що існуючі програмні рішення мають широкий набір функціональних можливостей для обліку замовлень, клієнтів, оплат, складу та звітності. Найбільш близькими до предметної області ремонтної майстерні є RO App / RemOnline та Orderry,

оскільки вони враховують специфіку сервісного обслуговування, роботу з ремонтними замовленнями, статусами та виконавцями.

Таблиця 1.1 – Порівняльна характеристика програмних рішень для обліку замовлень

Програмне рішення	Основне призначення	Переваги	Недоліки для ремонтної майстерні
RO App / RemOnline	Облік замовлень сервісних центрів і майстерень	Спеціалізація на сервісних процесах, облік клієнтів, замовлень, складу, оплат і звітів	Комерційна модель, надмірність окремих функцій для невеликої майстерні
Orderry	Управління ремонтними замовленнями та сервісним бізнесом	Повний цикл роботи із замовленням, статуси, відповідальні виконавці, історія змін	Обмежені можливості зміни логіки під власний навчальний проєкт
Odoo	Комплексна ERP-система для підприємств	Модульність, масштабованість, можливість розширення	Складність налаштування, надмірність функціоналу для простої майстерні
Zoho Inventory	Управління замовленнями, складом і продажами	Зручний веб-інтерфейс, облік товарів, замовлень, оплат і залишків	Орієнтація переважно на торгівлю, відсутність спеціалізованої логіки ремонту

Разом з тим готові системи мають певні недоліки для використання в межах кваліфікаційної роботи. Вони є комерційними продуктами, мають надлишковий функціонал, потребують налаштування та не завжди дозволяють повністю адаптувати логіку роботи під конкретну невелику майстерню. Крім того, використання готової системи не дає можливості продемонструвати повний цикл самостійного проєктування та розробки програмного забезпечення.

Отже, розробка власної веб-орієнтованої інформаційної системи обліку замовлень є доцільною. Така система може містити саме ті функції, які необхідні для роботи ремонтної майстерні: облік клієнтів, реєстрацію замовлень, збереження даних про пристрої, призначення майстрів, зміну статусів, фіксацію результатів діагностики та ремонту, облік використаних запчастин і формування звітної інформації для керівника.

1.4 Постановка задачі розробки інформаційної системи

У результаті аналізу діяльності ремонтної майстерні було встановлено, що процес обліку замовлень є одним із ключових елементів її роботи. Саме замовлення об’єднує інформацію про клієнта, пристрій, опис несправності, результати діагностики, виконані ремонтні роботи, використані запчастини, відповідального майстра, статус виконання та вартість послуг. За відсутності єдиної інформаційної системи ці дані можуть зберігатися у різних джерелах, що ускладнює пошук, контроль і подальше використання інформації.

Проведений аналіз процесу обліку замовлень показав, що для ефективної роботи ремонтної майстерні необхідно забезпечити централізоване збереження даних, зручний доступ до замовлень, контроль їх поточного стану, фіксацію результатів діагностики та ремонту, а також можливість перегляду історії звернень клієнтів. Особливо важливим є використання статусів замовлення, оскільки вони дозволяють відстежувати життєвий цикл кожного замовлення від моменту його створення до видачі відремонтованого пристрою клієнту.

Огляд існуючих програмних рішень показав, що на ринку вже існують системи для автоматизації сервісних центрів, ремонтних майстерень, складського обліку та роботи із замовленнями [5–8]. Такі системи мають широкий набір функцій, однак часто є комерційними, містять надмірний функціонал або потребують додаткового налаштування під конкретну предметну область. Для невеликої ремонтної майстерні доцільним є створення

простішої, але функціонально завершеної веб-орієнтованої системи, яка буде орієнтована саме на облік замовлень і контроль процесу ремонту.

Задачею кваліфікаційної роботи є розробка веб-орієнтованої інформаційної системи обліку замовлень ремонтної майстерні, яка забезпечує автоматизацію основних процесів приймання, супроводження, виконання та завершення замовлень.

Розроблювана інформаційна система повинна забезпечувати роботу з такими основними сутностями:

– користувачі системи;

– клієнти майстерні;

– пристрої, що передаються на ремонт;

– замовлення на ремонт;

– статуси замовлень;

– результати діагностики;

– виконані ремонтні роботи;

– використані запчастини;

– оплати;

– історія змін замовлення.

У системі доцільно передбачити декілька ролей користувачів: клієнта, адміністратора, майстра та керівника. Клієнт повинен мати можливість подати заявку та переглянути стан замовлення. Адміністратор повинен мати можливість створювати замовлення, реєструвати клієнтів, вносити дані про пристрій, призначати майстра та змінювати загальний статус замовлення. Майстер повинен працювати з призначеними йому замовленнями, вносити результати діагностики, опис виконаних робіт, інформацію про використані запчастини та оновлювати технічний статус ремонту. Керівник повинен мати доступ до перегляду всіх замовлень, контролю активних і завершених робіт, аналізу завантаженості майстрів і перегляду звітної інформації.

Основними функціональними вимогами до розроблюваної системи є:

– авторизація користувачів;

– створення, перегляд, редагування та видалення записів про клієнтів;

– створення, перегляд, редагування та закриття замовлень;

– збереження інформації про пристрій, який передається на ремонт;

– призначення відповідального майстра для виконання замовлення;

– зміна статусу замовлення відповідно до етапу його виконання;

– фіксація результатів діагностики;

– внесення інформації про виконані роботи;

– облік використаних запчастин;

– розрахунок або внесення вартості ремонту;

– пошук і фільтрація замовлень за номером, клієнтом, статусом, датою або майстром;

– перегляд історії ремонтних робіт;

– формування зведеної інформації для керівника.

Формування вимог є важливим етапом розробки програмного забезпечення, оскільки саме на цьому етапі визначаються функції системи, обмеження її роботи та очікувані результати використання [9]. Крім функціональних вимог, система повинна відповідати певним нефункціональним вимогам. Веб-інтерфейс має бути простим, зрозумілим і зручним для користувачів, які виконують різні ролі в майстерні. Дані повинні зберігатися в структурованому вигляді в базі даних. Система повинна забезпечувати коректну обробку введених даних, обмеження доступу відповідно до ролі користувача та можливість подальшого розширення функціональності.

Для реалізації інформаційної системи обрано клієнт-серверну архітектуру. Клієнтська частина має бути реалізована з використанням бібліотеки React, що дозволяє створити зручний веб-інтерфейс користувача [1]. Серверну частину доцільно реалізувати засобами мови програмування Python, яка забезпечує обробку запитів, взаємодію з базою даних і реалізацію основної бізнес-логіки [2]. Для збереження даних обрано реляційну систему

керування базами даних MySQL, яка підходить для структурованого обліку клієнтів, замовлень, пристроїв, ремонтних робіт і статусів [3].

У межах кваліфікаційної роботи необхідно виконати такі етапи розробки:

– сформувати вимоги до веб-орієнтованої інформаційної системи;

– розробити функціональну схему системи;

– спроєктувати архітектуру програмного забезпечення;

– розробити структуру бази даних;

– реалізувати серверну частину системи;

– реалізувати клієнтську частину системи;

– забезпечити взаємодію клієнтської частини з серверною через програмний інтерфейс;

– виконати тестування основних функцій системи;

– підготувати екранні форми та опис методики роботи користувача.

Очікуваним результатом виконання кваліфікаційної роботи є працездатна веб-орієнтована інформаційна система, яка дозволяє автоматизувати облік замовлень у ремонтній майстерні. Розроблена система повинна забезпечувати створення та супроводження замовлень, контроль їх статусів, облік клієнтів, пристроїв, ремонтних робіт і запчастин, а також надання зведеної інформації для керівника.

Таким чином, постановка задачі полягає у створенні програмного засобу, який поєднує простоту використання з достатнім набором функціональних можливостей для організації роботи ремонтної майстерні. Розробка такої системи дозволить підвищити впорядкованість обліку, зменшити кількість ручних операцій, прискорити пошук інформації та забезпечити контроль виконання замовлень на всіх етапах їх життєвого циклу.

2 ПРОЄКТУВАННЯ ВЕБ-ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ ЗАМОВЛЕНЬ

2.1 Визначення функціональних вимог до системи

На етапі проєктування веб-орієнтованої інформаційної системи обліку замовлень ремонтної майстерні необхідно визначити основні функціональні вимоги до програмного забезпечення. Функціональні вимоги описують дії, які повинна виконувати система, а також визначають можливості користувачів під час роботи з нею [9].

Розроблювана інформаційна система призначена для автоматизації процесів приймання, обліку, супроводження та завершення замовлень у ремонтній майстерні. Основним об’єктом обліку є замовлення на ремонт, яке містить інформацію про клієнта, пристрій, опис несправності, результати діагностики, виконані роботи, використані запчастини, відповідального майстра, поточний статус і вартість ремонту.

З урахуванням аналізу предметної області, виконаного у першому розділі, у системі доцільно передбачити такі ролі користувачів:

– клієнт;

– адміністратор;

– майстер;

– керівник.

Клієнт є зовнішнім користувачем системи. Він може подати заявку на ремонт, надати контактні дані, описати несправність пристрою, а також отримати інформацію про поточний стан замовлення. У спрощеній версії системи клієнт не обов’язково має повний особистий кабінет, однак для нього доцільно передбачити окремий інтерфейс подання заявки та перегляду статусу замовлення за його номером.

Адміністратор є внутрішнім користувачем системи та відповідає за первинне внесення даних. Він реєструє клієнтів, створює замовлення, вносить

інформацію про пристрій, зазначає опис несправності, призначає відповідального майстра та контролює зміну загального статусу замовлення.

Майстер працює із замовленнями, які були йому призначені. Він переглядає інформацію про пристрій і несправність, проводить діагностику, вносить результати перевірки, зазначає виконані ремонтні роботи, додає використані запчастини та оновлює технічний статус замовлення.

Керівник має доступ до перегляду загальної інформації про роботу майстерні. Він може переглядати активні та завершені замовлення, контролювати завантаженість майстрів, аналізувати кількість виконаних робіт і переглядати зведену інформацію.

Для забезпечення коректної роботи системи необхідно реалізувати механізм авторизації користувачів. Для внутрішніх користувачів, а саме адміністратора, майстра та керівника, авторизація повинна виконуватися за логіном і паролем. Після входу в систему користувач отримує доступ лише до тих функцій, які відповідають його ролі. Такий підхід відповідає принципам рольового керування доступом, за якого права користувача визначаються його роллю в системі [10]. Для клієнта може бути реалізований спрощений доступ: подання заявки без авторизації та перегляд статусу замовлення за номером замовлення або контактними даними. Такий підхід дозволяє не ускладнювати систему, але забезпечити зручну взаємодію клієнта з майстернею.

Основні функціональні вимоги до веб-орієнтованої інформаційної системи обліку замовлень наведено в таблиці 2.1.

Функціональні вимоги можна поділити на декілька груп: вимоги до взаємодії клієнта із системою, вимоги до авторизації внутрішніх користувачів, вимоги до роботи з клієнтами, вимоги до роботи із замовленнями, вимоги до процесу ремонту, вимоги до контролю статусів і вимоги до перегляду звітної інформації.

Вимоги до взаємодії клієнта із системою передбачають можливість подання заявки на ремонт і перегляду поточного статусу замовлення. Під час подання заявки клієнт повинен мати можливість вказати своє ім’я, номер

телефону, тип пристрою, модель пристрою та короткий опис несправності. Після обробки заявки адміністратором у системі створюється замовлення, яке надалі супроводжується працівниками майстерні. Перегляд статусу дозволяє клієнту отримувати базову інформацію про стан виконання ремонту без необхідності телефонувати до майстерні.

Таблиця 2.1 – Функціональні вимоги до інформаційної системи обліку замовлень

№	Функціональна вимога	Опис вимоги	Роль користувача
1	2	3	4
1	Подання заявки на ремонт	Система повинна дозволяти клієнту залишити заявку із зазначенням контактних даних, типу пристрою та опису несправності	Клієнт
2	Перегляд статусу замовлення клієнтом	Система повинна дозволяти клієнту отримати інформацію про поточний стан замовлення за його номером	Клієнт
3	Авторизація внутрішніх користувачів	Система повинна забезпечувати вхід адміністратора, майстра та керівника за логіном і паролем	Адміністратор, майстер, керівник
4	Керування клієнтами	Система повинна дозволяти створювати, переглядати, редагувати та видаляти записи про клієнтів	Адміністратор
5	Реєстрація пристрою	Система повинна дозволяти зберігати дані про пристрій, що передається на ремонт	Адміністратор
6	Створення замовлення	Система повинна забезпечувати створення нового замовлення із зазначенням клієнта, пристрою, несправності, дати приймання та початкового статусу	Адміністратор
7	Перегляд замовлень	Система повинна надавати можливість переглядати список замовлень з основною інформацією про них	Адміністратор, майстер, керівник

Продовження табл. 2.1

1	2	3	4
8	Редагування замовлення	Система повинна дозволяти змінювати дані замовлення відповідно до прав доступу користувача	Адміністратор, майстер
9	Призначення майстра	Система повинна дозволяти призначати відповідального майстра для виконання ремонтних робіт	Адміністратор
10	Фіксація результатів діагностики	Система повинна дозволяти майстру вносити результати первинної діагностики пристрою	Майстер
11	Облік виконаних робіт	Система повинна забезпечувати внесення інформації про виконані ремонтні роботи	Майстер
12	Облік використаних запчастин	Система повинна дозволяти зазначати запчастини, використані під час ремонту	Майстер
13	Зміна статусу замовлення	Система повинна підтримувати зміну статусу замовлення відповідно до етапу його виконання	Адміністратор, майстер
14	Розрахунок вартості ремонту	Система повинна дозволяти зберігати вартість робіт, запчастин і загальну суму до оплати	Адміністратор, майстер
15	Пошук і фільтрація замовлень	Система повинна забезпечувати пошук замовлень за номером, клієнтом, датою, статусом або майстром	Адміністратор, майстер, керівник
16	Перегляд історії ремонту	Система повинна зберігати історію виконаних робіт і змін статусу замовлення	Адміністратор, майстер, керівник
17	Формування звітної інформації	Система повинна надавати керівнику узагальнену інформацію про активні, завершені та скасовані замовлення	Керівник

До вимог авторизації належить забезпечення входу внутрішніх користувачів до системи за обліковими даними. Після авторизації система

повинна визначати роль користувача та надавати йому відповідний набір функцій. Наприклад, майстер не повинен мати доступу до керування всіма користувачами системи, а керівник може мати переважно права перегляду й аналізу інформації. Такий підхід відповідає принципу рольового розмежування доступу, коли дозволи користувача залежать від його ролі в системі [10].

Вимоги до роботи з клієнтами передбачають створення та збереження клієнтської бази. Для кожного клієнта необхідно зберігати прізвище, ім’я, номер телефону, електронну пошту за наявності, адресу або додаткові примітки. Наявність клієнтської бази дозволяє швидко знаходити попередні звернення клієнта та переглядати історію ремонтів.

Вимоги до роботи із замовленнями є основними для розроблюваної системи. Під час створення замовлення повинні фіксуватися дані клієнта, інформація про пристрій, опис несправності, дата приймання, відповідальний майстер, статус і попередні коментарі. Кожне замовлення повинно мати унікальний номер, за яким його можна швидко знайти в системі.

Вимоги до процесу ремонту пов’язані з діяльністю майстра. Система повинна дозволяти майстру переглядати призначені йому замовлення, вносити результати діагностики, описувати виконані роботи, зазначати використані запчастини та оновлювати технічний стан замовлення. Це дає змогу фіксувати не лише факт завершення ремонту, а й увесь процес виконання робіт.

Вимоги до контролю статусів передбачають підтримку життєвого циклу замовлення. Для системи доцільно передбачити такі статуси: «Нове замовлення», «Прийнято», «На діагностиці», «Узгодження вартості та строків», «В роботі», «Очікує запчастини», «Контроль якості», «Виконано», «Очікує оплату», «Видано клієнту», «Архів», «Скасовано». Використання статусів дозволяє швидко визначати поточний стан кожного замовлення.

Вимоги до звітної інформації пов’язані з потребами керівника майстерні. Система повинна забезпечувати перегляд кількості активних замовлень, завершених замовлень, скасованих замовлень, завантаженості майстрів і

загальної інформації про виконані роботи. У межах кваліфікаційної роботи достатньо реалізувати базовий рівень звітності у вигляді зведених показників або таблиць.

Функціональні вимоги до програмного забезпечення визначають основні операції, які повинна виконувати система, та є основою для подальшого проєктування її структури й алгоритмів роботи [9]. Узагальнену структуру функціональних можливостей системи представлено у вигляді схеми функціональних модулів, наведеної на рисунку 2.1.

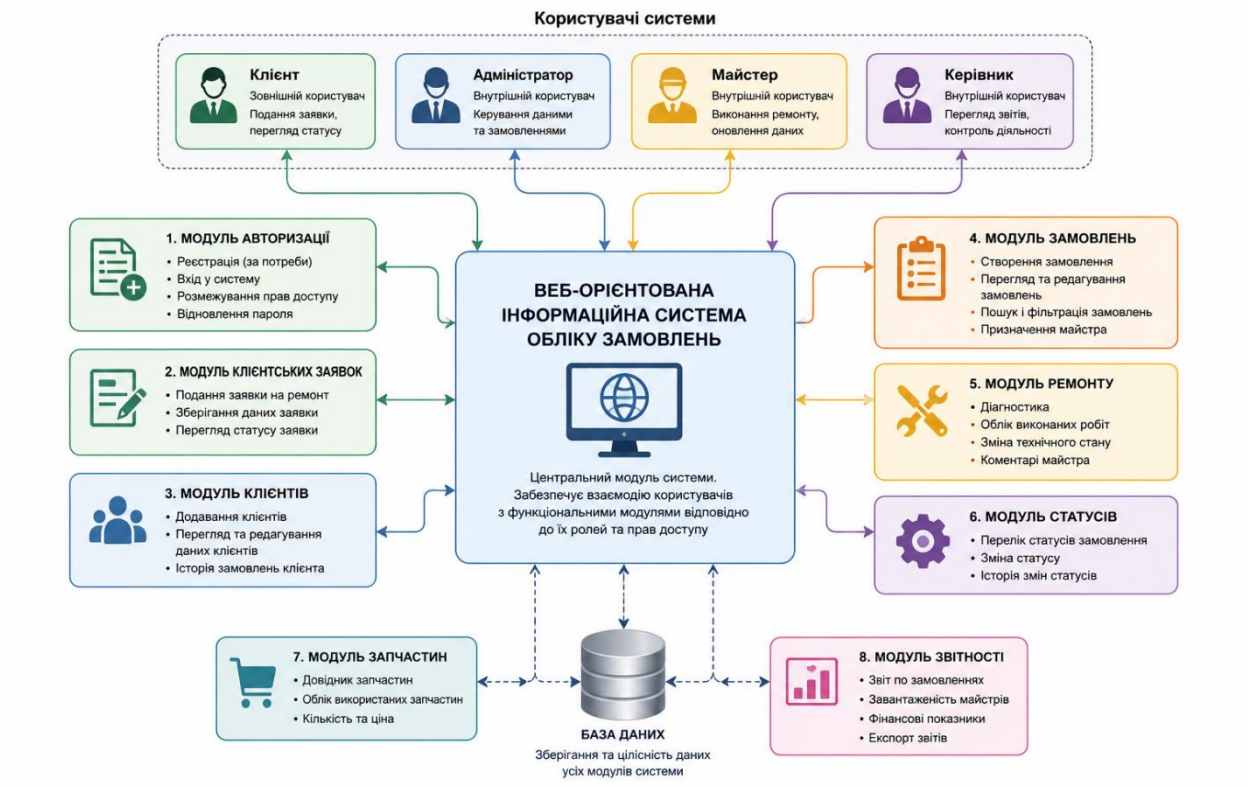

Рисунок 2.1 – Схема функціональних модулів інформаційної системи обліку замовлень

На рисунку 2.1 відображено основні модулі системи: модуль авторизації, модуль клієнтських заявок, модуль клієнтів, модуль замовлень, модуль ремонту, модуль статусів, модуль запчастин і модуль звітності. Центральним елементом схеми є веб-орієнтована інформаційна система, яка

взаємодіє з базою даних і забезпечує роботу користувачів відповідно до їх ролей.

Окрім функціональних вимог, під час проєктування необхідно враховувати обмеження, пов’язані зі зручністю використання системи. Інтерфейс повинен бути простим і зрозумілим, оскільки користувачами системи можуть бути як працівники майстерні, так і клієнти. Основні операції, такі як подання заявки, створення замовлення, пошук замовлення, зміна статусу та внесення результатів ремонту, мають виконуватися швидко й без зайвих дій. Зручність і зрозумілість інтерфейсу є важливими характеристиками якості програмного продукту, оскільки вони впливають на ефективність роботи користувача із системою [11].

Таким чином, визначені функціональні вимоги формують основу для подальшого проєктування архітектури, бази даних і користувацького інтерфейсу інформаційної системи. Реалізація зазначених вимог дозволить створити просту, але достатньо функціональну веб-орієнтовану систему, яка забезпечить облік замовлень ремонтної майстерні, взаємодію клієнта з майстернею та контроль виконання замовлень на всіх основних етапах.

2.2 Розробка функціональної схеми системи

Функціональна схема веб-орієнтованої інформаційної системи обліку замовлень ремонтної майстерні відображає склад основних модулів системи, взаємозв’язки між ними, ролі користувачів та напрямки обміну даними. Розробка такої схеми є необхідним етапом проєктування, оскільки дозволяє визначити загальну логіку роботи програмного забезпечення до початку його реалізації [9].

Розроблювана система призначена для автоматизації основних процесів ремонтної майстерні: подання заявки клієнтом, реєстрації клієнта, створення замовлення, внесення інформації про пристрій, призначення майстра, фіксації

результатів діагностики, обліку виконаних робіт, зміни статусів замовлення, обліку використаних запчастин і формування звітної інформації для керівника.

З урахуванням визначених функціональних вимог система повинна забезпечувати взаємодію таких користувачів:

– клієнта, який подає заявку на ремонт і переглядає статус замовлення;

– адміністратора, який реєструє клієнтів, створює замовлення та призначає майстра;

– майстра, який виконує діагностику, ремонт і вносить технічну інформацію;

– керівника, який контролює роботу майстерні та переглядає звітну інформацію.

Основою функціональної структури системи є центральний програмний модуль, який забезпечує взаємодію між клієнтським інтерфейсом, серверною частиною та базою даних. Клієнтська частина системи реалізує веб-інтерфейс для роботи користувачів, серверна частина обробляє запити та виконує бізнес-логіку, а база даних забезпечує збереження інформації про клієнтів, користувачів, замовлення, пристрої, статуси, запчастини та історію ремонтів.

Загальну функціональну схему веб-орієнтованої інформаційної системи обліку замовлень наведено на рисунку 2.2.

На рисунку 2.2 зображено чотири групи користувачів, веб-інтерфейс системи, серверну частину, функціональні модулі та базу даних. Клієнт взаємодіє із системою через сторінку подання заявки та сторінку перегляду статусу замовлення. Адміністратор, майстер і керівник працюють із системою через авторизований інтерфейс. Усі дії користувачів передаються на серверну частину, яка обробляє запити та звертається до бази даних. Розмежування доступу користувачів відповідно до їхніх ролей дозволяє надати кожній групі лише ті функції, які потрібні для виконання її завдань [10].

До складу функціональної схеми системи входять такі основні модулі:

– модуль авторизації;

– модуль клієнтських заявок;

– модуль клієнтів;

– модуль замовлень;

– модуль ремонту;

– модуль статусів;

– модуль запчастин;

– модуль звітності.

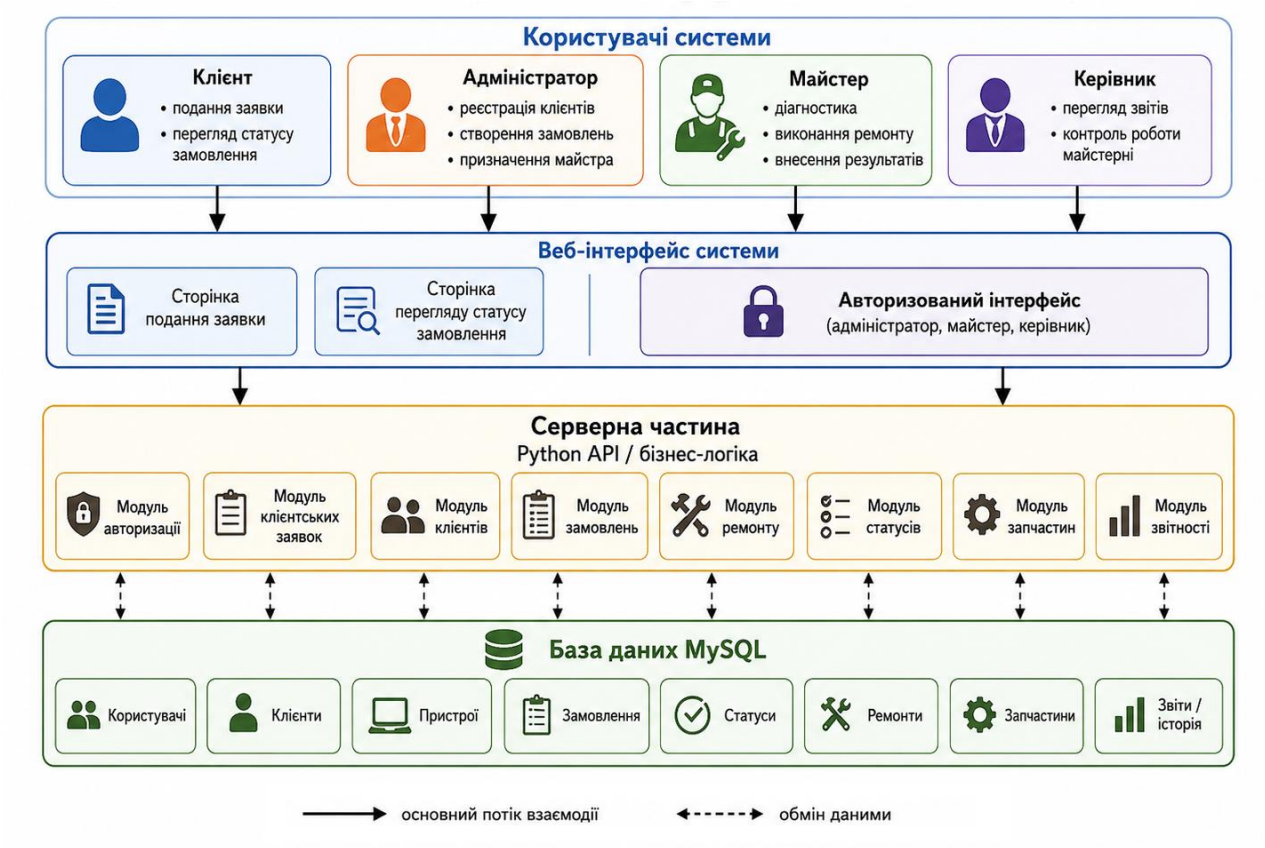

Рисунок 2.2 – Функціональна схема веб-орієнтованої інформаційної системи обліку замовлень

Модуль авторизації забезпечує вхід внутрішніх користувачів до системи. До внутрішніх користувачів належать адміністратор, майстер і керівник. Після успішної авторизації система визначає роль користувача та надає доступ до відповідних функцій. Наприклад, адміністратор має доступ до створення замовлень і керування клієнтами, майстер — до призначених йому замовлень, а керівник — до перегляду звітів і статистичної інформації.

Модуль клієнтських заявок призначений для приймання первинних звернень від клієнтів. За допомогою цього модуля клієнт може залишити заявку на ремонт, зазначивши своє ім’я, номер телефону, тип пристрою, модель та опис несправності. Після надходження заявки адміністратор перевіряє її та створює на її основі повноцінне замовлення.

Модуль клієнтів забезпечує збереження та обробку інформації про клієнтів майстерні. У цьому модулі зберігаються контактні дані клієнта, історія його звернень і пов’язані з ним замовлення. Наявність такого модуля дозволяє швидко знаходити клієнта під час повторного звернення та переглядати історію виконаних ремонтів.

Модуль замовлень є центральним модулем інформаційної системи. Він забезпечує створення, перегляд, редагування, пошук і закриття замовлень. У замовленні зберігаються дані про клієнта, пристрій, опис несправності, дату створення, відповідального майстра, поточний статус, вартість ремонту та примітки. Саме цей модуль об’єднує інформацію, яка надходить з інших функціональних частин системи.

Модуль ремонту використовується майстром для внесення технічної інформації про виконання робіт. У цьому модулі фіксуються результати діагностики, опис виконаних ремонтних операцій, використані запчастини, коментарі майстра та рекомендації щодо подальшої експлуатації пристрою. Модуль ремонту дозволяє зберігати не лише кінцевий результат, а й хід виконання замовлення.

Модуль статусів забезпечує керування життєвим циклом замовлення. У системі передбачено такі основні статуси: «Нове замовлення», «Прийнято», «На діагностиці», «Узгодження вартості та строків», «В роботі», «Очікує запчастини», «Контроль якості», «Виконано», «Очікує оплату», «Видано клієнту», «Архів», «Скасовано». Використання статусів дозволяє користувачам швидко визначити поточний стан замовлення та контролювати його проходження через усі етапи ремонту.

Модуль запчастин призначений для обліку комплектуючих, які можуть використовуватися під час ремонту. У межах розроблюваної системи цей модуль може містити найменування запчастини, кількість, вартість і прив’язку до конкретного замовлення. Такий підхід дозволяє враховувати використані матеріали під час розрахунку загальної вартості ремонту.

Модуль звітності призначений для керівника майстерні. Він забезпечує перегляд узагальненої інформації про кількість активних, завершених і скасованих замовлень, завантаженість майстрів, виконані роботи та фінансові показники. У межах кваліфікаційної роботи достатньо реалізувати базову звітність у вигляді таблиць і зведених показників. Відображення інформації у зрозумілій формі є важливим для швидкого аналізу стану роботи майстерні та прийняття управлінських рішень [11].

Відповідність функціональних модулів системи основним діям користувачів наведено в таблиці 2.2.

Особливістю розроблюваної системи є те, що вона має підтримувати як зовнішню, так і внутрішню взаємодію. Зовнішня взаємодія пов’язана з клієнтом, який може подати заявку на ремонт і перевірити стан замовлення. Внутрішня взаємодія охоплює роботу працівників майстерні: адміністратора, майстра та керівника. Для кожного типу користувача система повинна надавати окремий набір функцій відповідно до його ролі.

Для кращого відображення взаємодії користувачів із функціональними модулями системи розроблено схему розподілу функцій за ролями, наведену на рисунку 2.3.

На рисунку 2.3 показано, які модулі доступні кожній ролі. Клієнту доступні модуль клієнтських заявок і перегляд статусу замовлення. Адміністратор працює з модулями клієнтів, замовлень, статусів і частково із запчастинами. Майстер використовує модулі замовлень, ремонту, статусів і запчастин. Керівник має доступ до модуля замовлень у режимі перегляду та до модуля звітності.

Таблиця 2.2 – Відповідність функціональних модулів системи діям користувачів

Функціональний модуль	Основні дії	Користувачі
Модуль авторизації	Вхід у систему, перевірка логіна і пароля, визначення ролі користувача	Адміністратор, майстер, керівник
Модуль клієнтських заявок	Подання заявки, перегляд статусу заявки або замовлення	Клієнт
Модуль клієнтів	Додавання, перегляд, редагування даних клієнта, перегляд історії звернень	Адміністратор
Модуль замовлень	Створення, перегляд, редагування, пошук і закриття замовлень	Адміністратор, майстер, керівник
Модуль ремонту	Внесення результатів діагностики, опис виконаних робіт, додавання коментарів	Майстер
Модуль статусів	Зміна статусу замовлення, перегляд історії зміни статусів	Адміністратор, майстер
Модуль запчастин	Додавання використаних запчастин до замовлення, облік вартості запчастин	Майстер, адміністратор
Модуль звітності	Перегляд зведених показників, аналіз активних і завершених замовлень	Керівник

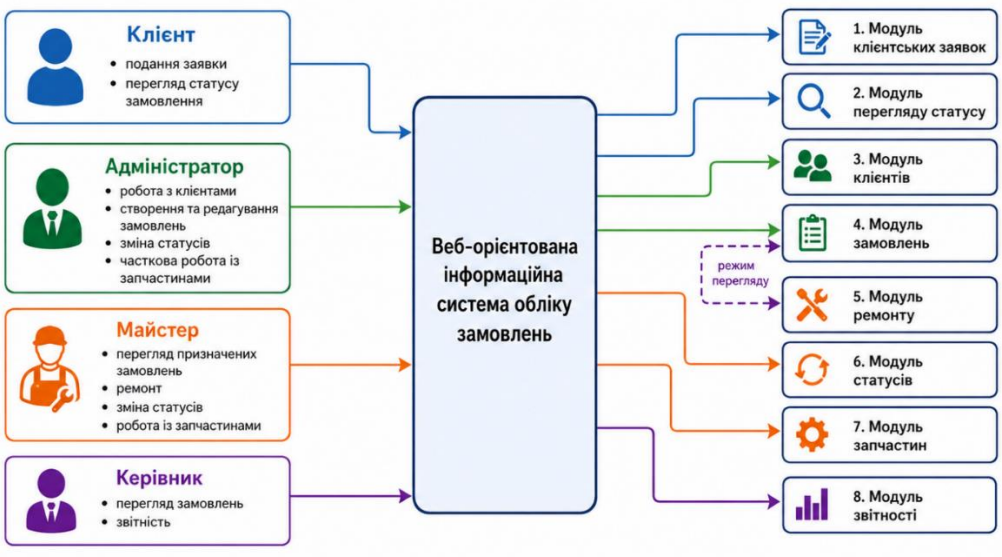

Рисунок 2.3 – Схема розподілу функцій системи за ролями користувачів

Під час роботи системи основний інформаційний потік формується навколо замовлення. Спочатку клієнт подає заявку або звертається до майстерні. Далі адміністратор створює замовлення, додає інформацію про клієнта та пристрій, після чого призначає майстра. Майстер проводить діагностику, виконує ремонт, зазначає використані запчастини та змінює статус замовлення. Після завершення ремонту адміністратор або майстер переводить замовлення до стану готовності, а після оплати та видачі пристрою клієнту замовлення закривається.

Схему інформаційного потоку під час обробки замовлення наведено на рисунку 2.4.

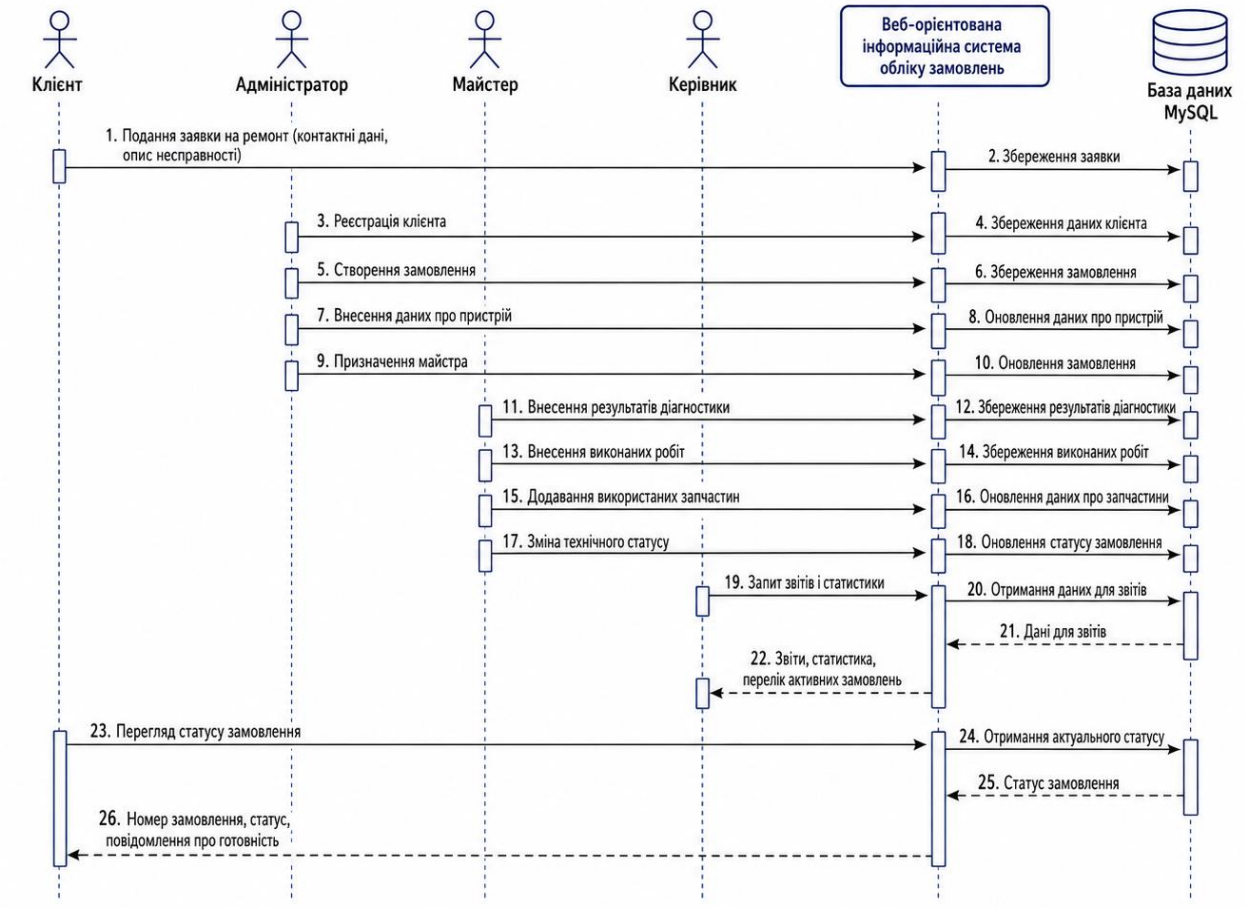

Рисунок 2.4 – Схема інформаційного потоку обробки замовлення

На рисунку 2.4 відображено послідовність передачі даних між клієнтом, адміністратором, майстром, керівником і базою даних. Схема повинна

показувати, що кожна дія користувача супроводжується збереженням або оновленням інформації в базі даних. Це забезпечує актуальність даних і можливість перегляду історії замовлення.

Функціональна схема системи також повинна враховувати особливості обраної технологічної реалізації. Оскільки клієнтська частина системи розробляється з використанням React, вона відповідає за відображення сторінок, форм, таблиць і кнопок керування [1]. Серверна частина, реалізована засобами Python, відповідає за обробку запитів, перевірку даних, виконання бізнес-логіки та взаємодію з базою даних [2]. База даних MySQL забезпечує централізоване збереження інформації та підтримує зв’язки між основними сутностями системи [3].

У загальному вигляді взаємодія програмних компонентів системи відбувається таким чином: користувач виконує дію у веб-інтерфейсі, клієнтська частина надсилає запит до серверної частини, сервер обробляє запит, звертається до бази даних, отримує або змінює необхідні дані та повертає результат на клієнтську частину. Така схема відповідає клієнт-серверному підходу та дозволяє розділити інтерфейс користувача, бізнес-логіку і збереження даних [12].

Таким чином, розроблена функціональна схема системи дозволяє визначити склад основних модулів, розподіл функцій між користувачами та порядок обміну даними між компонентами. Запропонована структура є достатньо простою для реалізації в межах кваліфікаційної роботи, але водночас охоплює основні потреби ремонтної майстерні щодо обліку клієнтів, замовлень, пристроїв, ремонтних робіт, статусів, запчастин і звітної інформації.

2.3 Розробка алгоритму роботи системи

Після визначення функціональних вимог і розробки функціональної схеми веб-орієнтованої інформаційної системи обліку замовлень необхідно

описати алгоритмічну побудову роботи системи. Алгоритм роботи системи визначає послідовність дій користувачів, порядок обробки запитів, взаємодію клієнтської частини із серверною частиною та базою даних.

Розроблювана система має клієнт-серверну архітектуру. Користувач взаємодіє із системою через веб-інтерфейс, реалізований засобами React [1]. Після виконання дії у веб-інтерфейсі формується HTTP-запит до серверної частини [13]. Серверна частина, реалізована засобами Python, приймає запит, перевіряє отримані дані, виконує необхідну бізнес-логіку, звертається до бази даних MySQL і повертає результат на клієнтську частину [2; 3]. Клієнтська частина відображає користувачу результат виконаної операції. Такий порядок взаємодії відповідає клієнт-серверному підходу, за якого інтерфейс користувача, логіка обробки та збереження даних розділені між окремими компонентами [12].

Загальна алгоритмічна побудова роботи системи складається з таких етапів:

– відкриття користувачем веб-сторінки системи;

– вибір режиму роботи залежно від ролі користувача;

– авторизація внутрішнього користувача або подання заявки клієнтом;

– введення або перегляд даних у веб-інтерфейсі;

– передача запиту на серверну частину;

– перевірка коректності отриманих даних;

– виконання відповідної операції в системі;

– збереження або отримання даних із бази даних;

– повернення результату на клієнтську частину;

– відображення повідомлення, таблиці, форми або статусу користувачу.

У загальному вигляді алгоритм роботи веб-орієнтованої інформаційної системи обліку замовлень наведено на рисунку 2.5.

На рисунку 2.5 показано початок роботи системи, вибір типу користувача, перевірку необхідності авторизації, виконання дій відповідно до ролі, звернення до серверної частини, роботу з базою даних і завершення

операції. Така схема дозволяє узагальнено представити логіку функціонування системи незалежно від конкретного модуля.

Одним із ключових алгоритмів системи є алгоритм авторизації внутрішнього користувача. До внутрішніх користувачів належать адміністратор, майстер і керівник. Авторизація необхідна для обмеження доступу до функцій системи відповідно до ролі користувача [10].

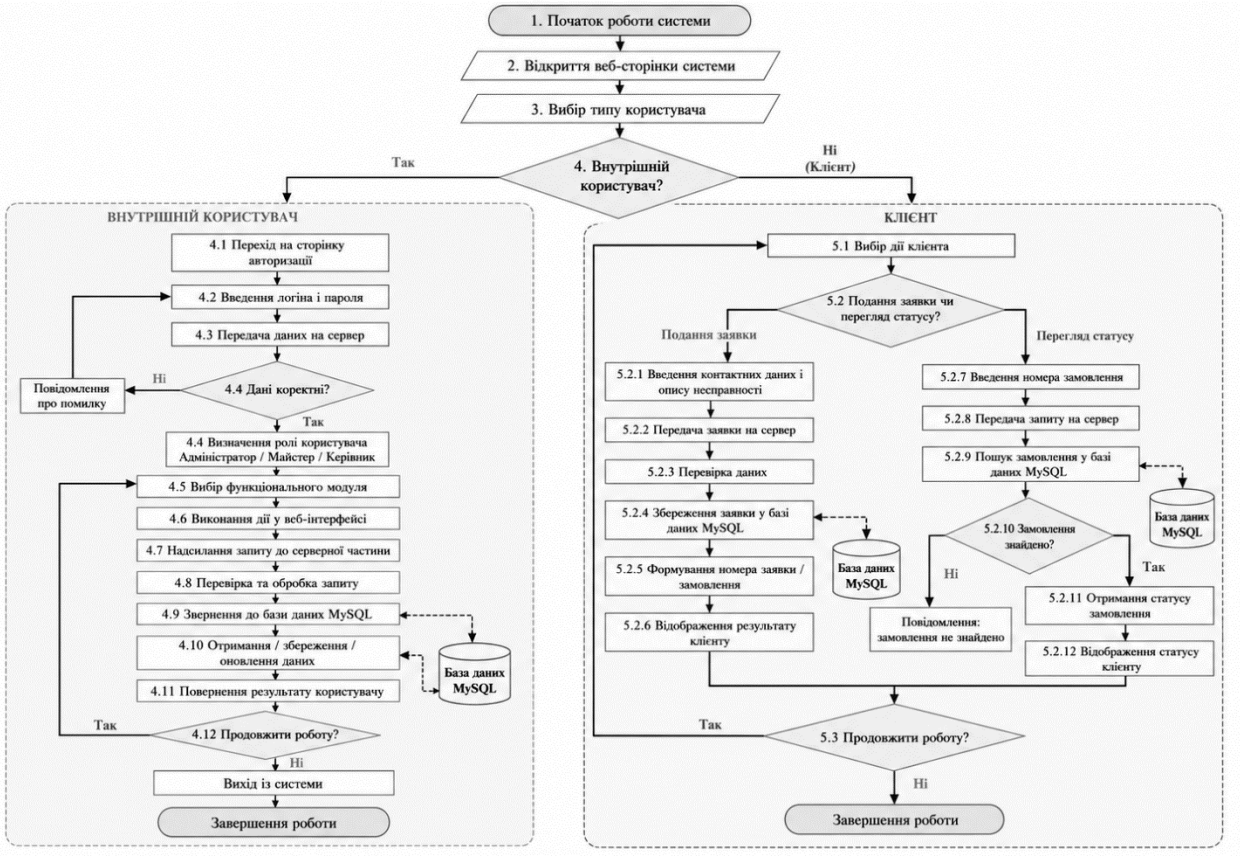

Рисунок 2.5 – Загальний алгоритм роботи веб-орієнтованої інформаційної системи обліку замовлень

Алгоритм авторизації користувача виконується таким чином. Користувач відкриває сторінку входу та вводить логін і пароль. Клієнтська частина передає ці дані на сервер. Сервер перевіряє, чи існує користувач із такими обліковими даними в базі даних. Якщо дані некоректні, система виводить повідомлення про помилку. Якщо дані правильні, сервер визначає роль користувача, створює сеанс роботи або токен доступу та повертає

клієнтській частині інформацію про успішний вхід. Після цього користувач переходить до відповідної панелі керування.

Алгоритм авторизації користувача наведено на рисунку 2.6.

Основним алгоритмом для адміністратора є алгоритм створення нового замовлення. Цей алгоритм використовується під час приймання пристрою на ремонт або під час обробки заявки, яку клієнт подав через веб-форму.

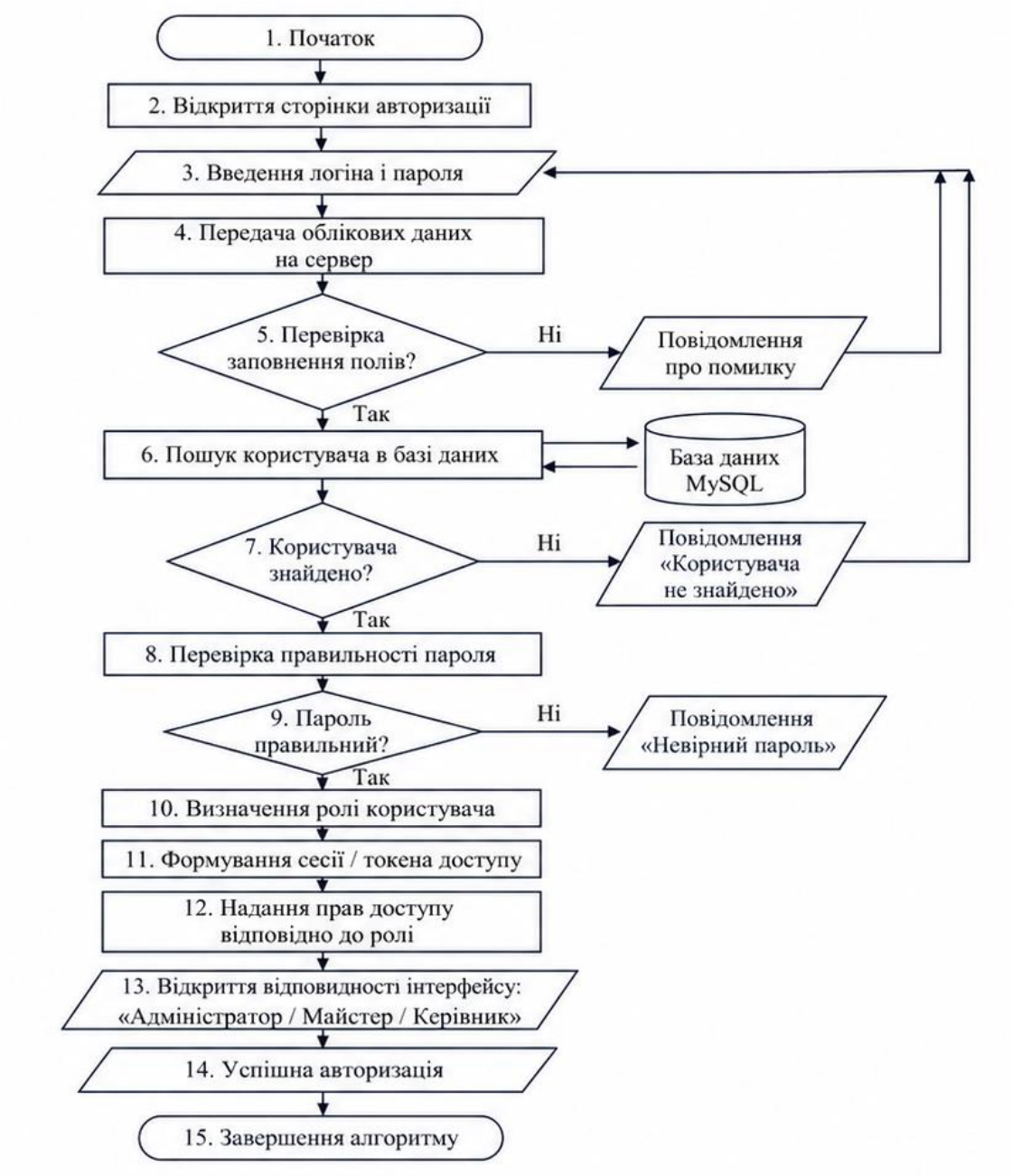

Рисунок 2.6 – Алгоритм авторизації внутрішнього користувача системи

Алгоритм створення замовлення починається з вибору або реєстрації клієнта. Якщо клієнт уже існує в базі даних, адміністратор обирає його зі списку. Якщо клієнт звернувся вперше, адміністратор створює новий запис про клієнта. Далі в систему вносяться дані про пристрій: тип, модель, серійний номер за наявності, зовнішній стан і комплектація. Після цього адміністратор зазначає опис несправності, дату приймання, початковий статус замовлення та призначає відповідального майстра. Після перевірки заповнених полів система зберігає замовлення в базі даних і присвоює йому унікальний номер.

Алгоритм створення нового замовлення наведено на рисунку 2.7.

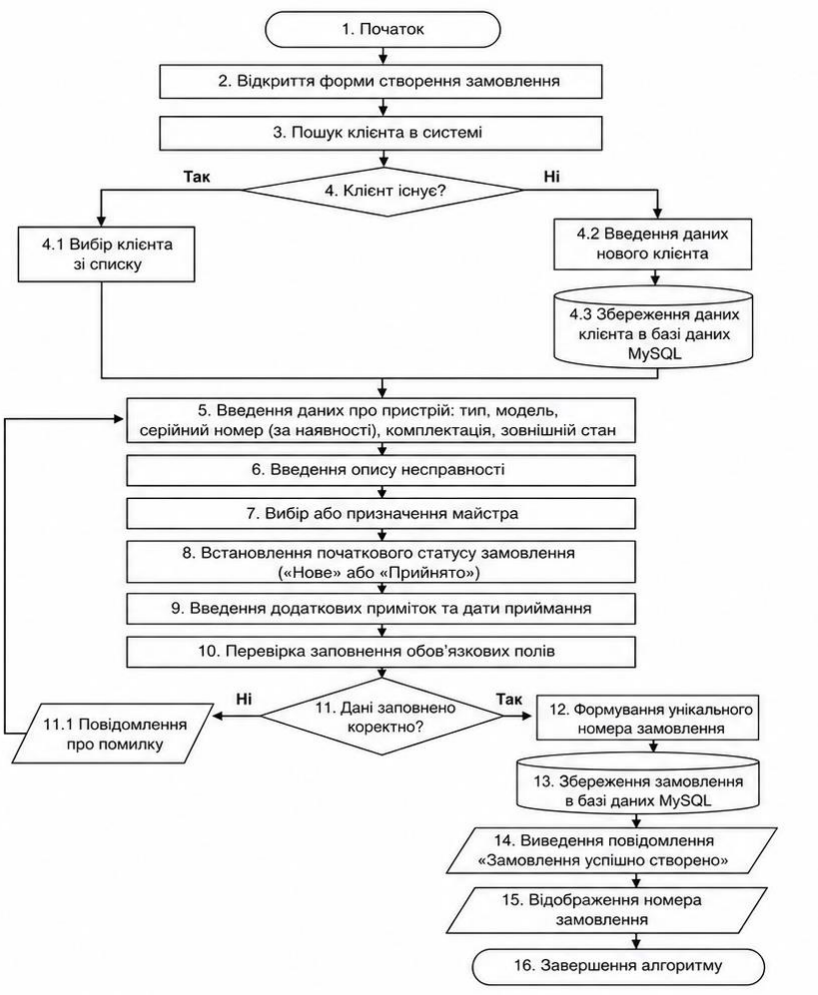

Рисунок 2.7 – Алгоритм створення нового замовлення

Для майстра ключовим є алгоритм обробки призначеного замовлення. Він охоплює перегляд замовлення, проведення діагностики, внесення

результатів перевірки, опис виконаних робіт, додавання використаних запчастин і зміну технічного статусу.

Після авторизації майстер переходить до списку призначених йому замовлень. Він обирає потрібне замовлення та переглядає інформацію про клієнта, пристрій і заявлену несправність. Далі майстер проводить діагностику та вносить її результати до системи. Якщо ремонт неможливий або недоцільний, замовлення може бути переведене у статус «Скасовано» або повернуте адміністратору для узгодження з клієнтом. Якщо ремонт можливий, майстер виконує роботи, додає інформацію про використані запчастини, описує виконані операції та змінює статус замовлення. Після завершення ремонту замовлення переходить до етапу контролю якості або до статусу «Виконано».

Алгоритм обробки замовлення майстром наведено на рисунку 2.8.

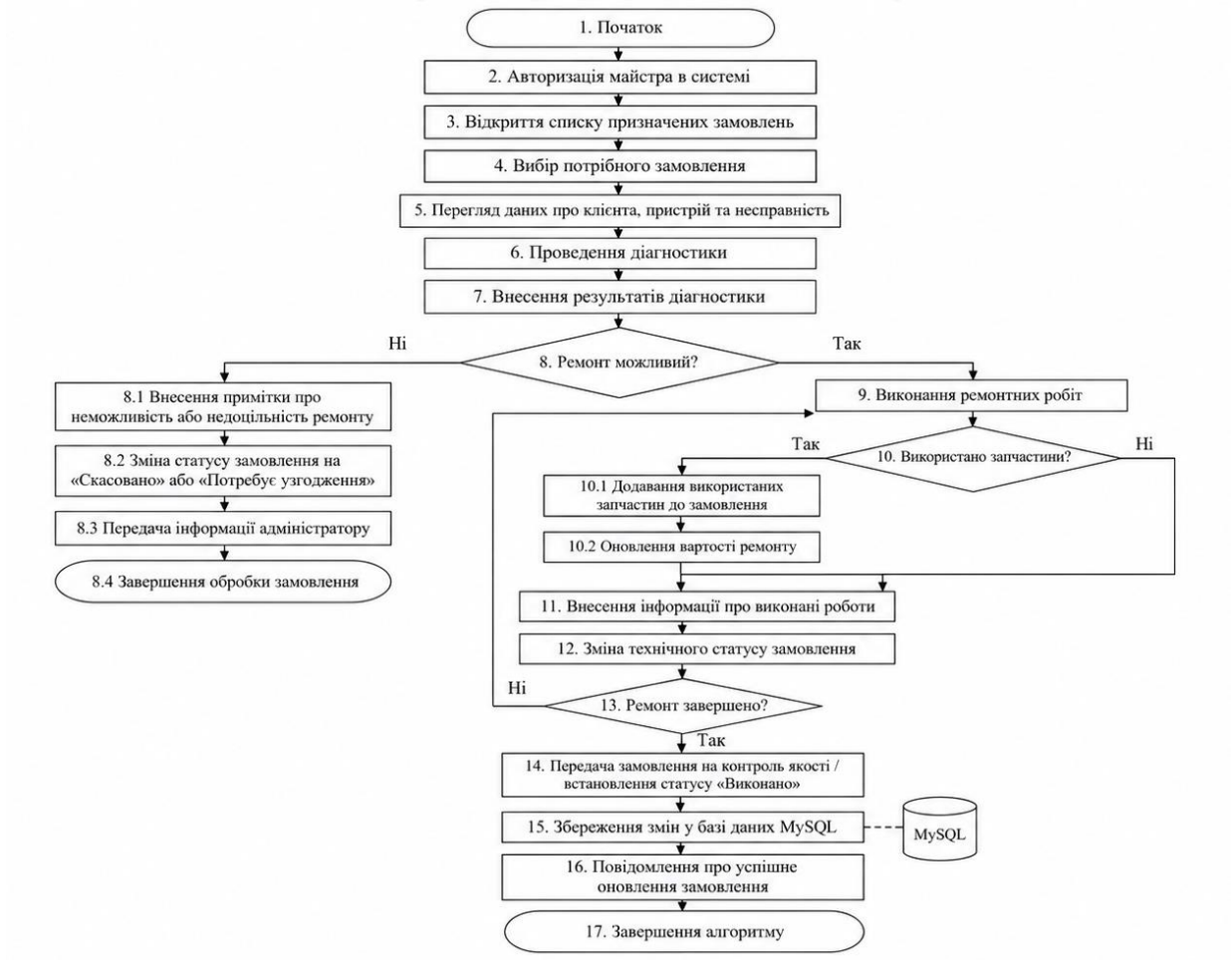

Рисунок 2.8 – Алгоритм обробки замовлення майстром

Окремо слід розглянути алгоритм перегляду статусу замовлення клієнтом. Цей алгоритм є важливим для зручності взаємодії клієнта з майстернею, оскільки дозволяє отримати інформацію про стан ремонту без звернення до адміністратора.

Клієнт відкриває сторінку перегляду статусу замовлення та вводить номер замовлення. За потреби система може додатково вимагати номер телефону для перевірки належності замовлення клієнту. Після введення даних клієнтська частина передає запит на сервер. Сервер перевіряє наявність замовлення в базі даних. Якщо замовлення не знайдено, користувачу виводиться повідомлення про помилку. Якщо замовлення знайдено, система повертає клієнту поточний статус, короткий опис етапу виконання та повідомлення про готовність у разі завершення ремонту.

Алгоритм перегляду статусу замовлення клієнтом наведено на рисунку 2.9.

Для керівника основним є алгоритм перегляду звітної інформації. Після авторизації керівник переходить до розділу звітності, обирає період або тип звіту, після чого система формує запит до бази даних. На основі отриманих даних система відображає кількість активних, завершених і скасованих замовлень, а також завантаженість майстрів. У межах кваліфікаційної роботи такий алгоритм може бути реалізований у вигляді перегляду зведеної таблиці або інформаційної панелі.

Зведений перелік основних алгоритмів системи наведено в таблиці 2.3.

Під час розробки алгоритмів важливо враховувати перевірку введених даних. Наприклад, під час створення замовлення система повинна перевіряти наявність обов’язкових полів: імені клієнта, номера телефону, типу пристрою, опису несправності та початкового статусу. Під час авторизації перевіряється правильність логіна та пароля. Під час зміни статусу замовлення перевіряється допустимість переходу між станами.

Також важливим є збереження історії змін. Кожна суттєва дія із замовленням, наприклад створення, призначення майстра, внесення

результатів діагностики, зміна статусу або додавання запчастин, повинна супроводжуватися оновленням даних у базі MySQL. Це дозволяє зберігати актуальний стан замовлення та, за потреби, переглядати історію його виконання.

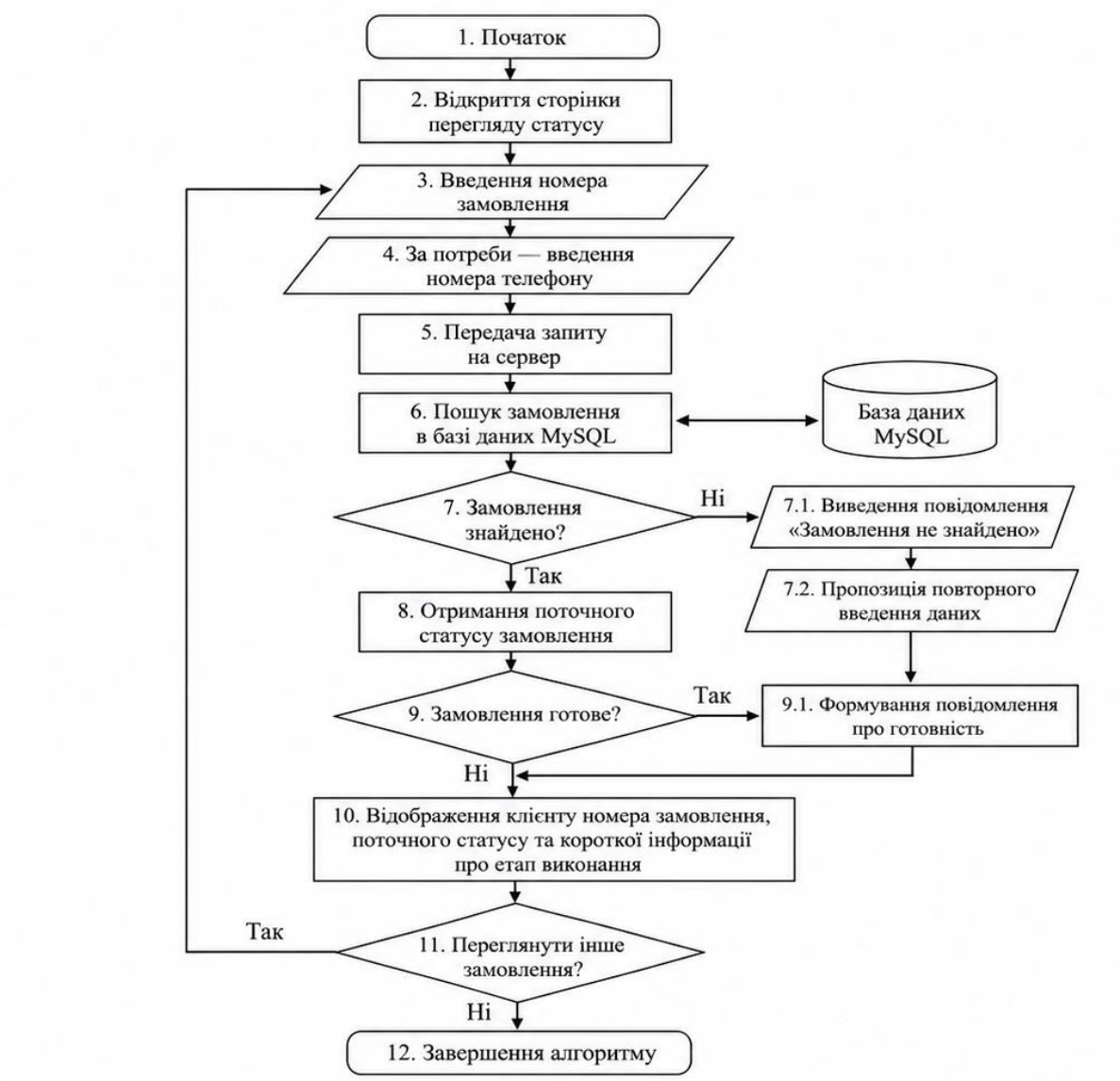

Рисунок 2.9 – Алгоритм перегляду статусу замовлення клієнтом

Алгоритмічна побудова системи повинна бути простою та зрозумілою, оскільки розроблювана система орієнтована на використання в невеликій ремонтній майстерні. Надмірне ускладнення логіки може негативно вплинути на зручність використання. Тому в межах кваліфікаційної роботи доцільно реалізувати базові, але завершені алгоритми, які охоплюють основні процеси:

створення замовлення, супроводження ремонту, зміну статусів, перегляд інформації клієнтом і контроль роботи керівником.

Таблиця 2.3 – Основні алгоритми роботи інформаційної системи

№	Назва алгоритму	Основна роль користувача	Результат виконання
1	Загальний алгоритм роботи системи	Усі користувачі	Виконання дії відповідно до ролі користувача
2	Алгоритм авторизації внутрішнього користувача	Адміністратор, майстер, керівник	Надання доступу до відповідного інтерфейсу
3	Алгоритм створення нового замовлення	Адміністратор	Створене замовлення з унікальним номером
4	Алгоритм обробки замовлення майстром	Майстер	Оновлене замовлення з результатами діагностики або ремонту
5	Алгоритм перегляду статусу замовлення	Клієнт	Відображення поточного статусу замовлення
6	Алгоритм перегляду звітної інформації	Керівник	Отримання зведених показників роботи майстерні

Таким чином, розроблені алгоритми роботи системи визначають порядок взаємодії користувачів із веб-орієнтованою інформаційною системою, послідовність обробки запитів і правила оновлення даних у базі MySQL. Подальше проєктування архітектури, бази даних і користувацького інтерфейсу має виконуватися з урахуванням описаних алгоритмів.

2.4 Проєктування архітектури програмного забезпечення

Архітектура програмного забезпечення визначає склад основних компонентів системи, їх функції та порядок взаємодії. Для веб-орієнтованої інформаційної системи обліку замовлень ремонтної майстерні обрано клієнт-

серверну багаторівневу архітектуру, яка дозволяє розділити рівень представлення, рівень прикладної логіки та рівень збереження даних [12; 14].

До складу системи входять:

– клієнтська частина, реалізована засобами React [1];

– серверна частина, реалізована мовою Python [2];

– програмний інтерфейс взаємодії між компонентами;

– база даних MySQL [3].

Користувачі працюють із системою через веббраузер. Клієнтська частина відповідає за відображення інтерфейсу, введення даних і формування запитів. Серверна частина виконує перевірку даних, реалізує бізнес-логіку, контролює права доступу та взаємодіє з базою даних. MySQL забезпечує централізоване збереження інформації про користувачів, клієнтів, пристрої, замовлення, ремонтні роботи, статуси, запчастини та історію змін.

Загальну архітектуру системи наведено на рисунку 2.10.

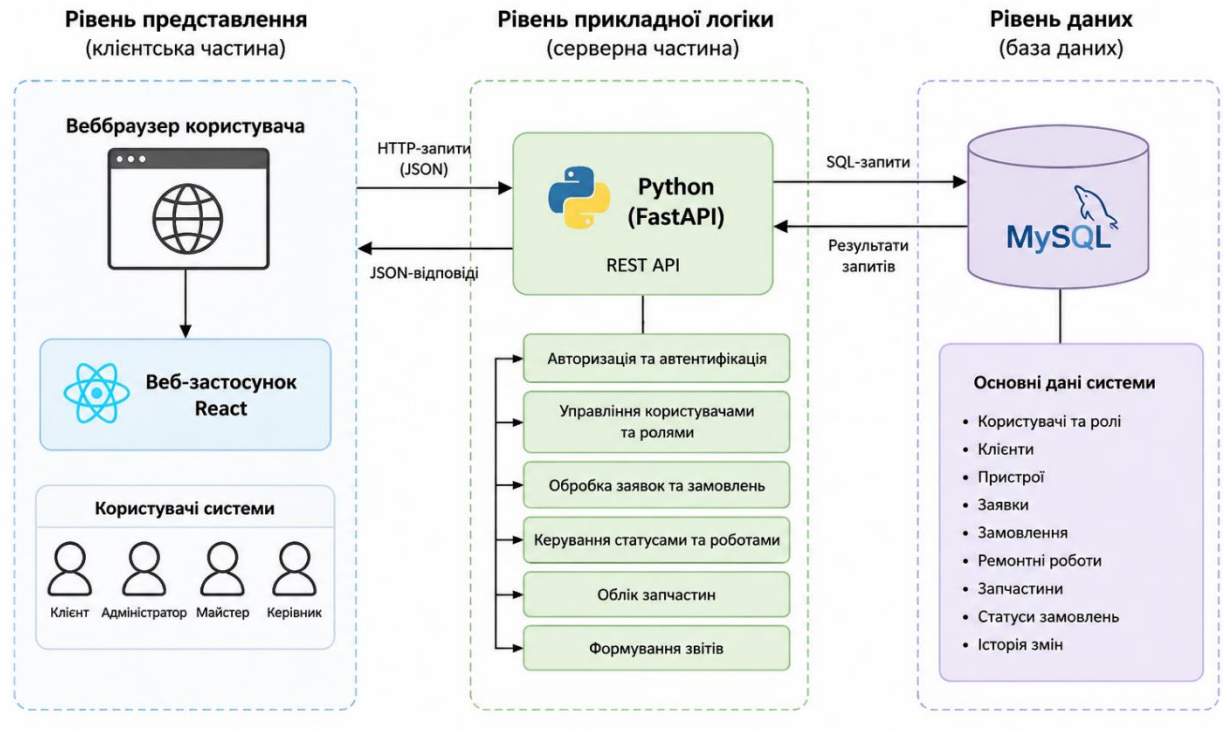

Рисунок 2.10 – Загальна архітектура веб-орієнтованої інформаційної системи обліку замовлень

Рівень представлення реалізується як вебзастосунок React. Він містить сторінки подання заявки, перегляду статусу замовлення, авторизації, роботи з клієнтами, замовленнями, ремонтом і звітністю. Склад доступних сторінок залежить від ролі користувача.

Рівень прикладної логіки реалізується серверною частиною Python із використанням FastAPI. Він забезпечує авторизацію користувачів, перевірку прав доступу, обробку заявок, створення та редагування замовлень, зміну статусів, облік виконаних робіт і формування звітної інформації.

Рівень даних представлений базою даних MySQL. У ній зберігаються облікові записи користувачів, дані клієнтів, відомості про пристрої, заявки, замовлення, результати діагностики, виконані роботи, використані запчастини, статуси та історія їх змін.

Взаємодія між клієнтською та серверною частинами здійснюється через REST API. Дані передаються у форматі JSON, а для виконання операцій використовуються HTTP-методи GET, POST, PATCH і DELETE.

Основні маршрути програмного інтерфейсу наведено в таблиці 2.4.

Таблиця 2.4 – Основні маршрути програмного інтерфейсу

Метод	Маршрут	Призначення
POST	/api/auth/login	Авторизація користувача
POST	/api/requests	Подання клієнтської заявки
GET	/api/orders/status/{number}	Перегляд статусу замовлення
GET	/api/clients	Отримання списку клієнтів
POST	/api/clients	Створення клієнта
GET	/api/orders	Отримання списку замовлень
POST	/api/orders	Створення замовлення
PATCH	/api/orders/{id}	Оновлення замовлення
PATCH	/api/orders/{id}/status	Зміна статусу
POST	/api/orders/{id}/diagnostics	Додавання результатів діагностики
POST	/api/orders/{id}/works	Додавання виконаних робіт
GET	/api/reports/summary	Отримання зведеної звітності

Для внутрішніх користувачів передбачається авторизація за логіном і паролем. Після успішної перевірки сервер визначає роль користувача та надає відповідний рівень доступу. Адміністратор працює з клієнтами й замовленнями, майстер — із призначеними ремонтами, керівник — зі звітами та загальною інформацією.

Типова послідовність виконання операції має такий вигляд: користувач виконує дію у вебінтерфейсі, React надсилає запит до серверної частини, сервер перевіряє дані та права доступу, звертається до MySQL і повертає результат користувачу.

Схему взаємодії програмних компонентів наведено на рисунку 2.11.

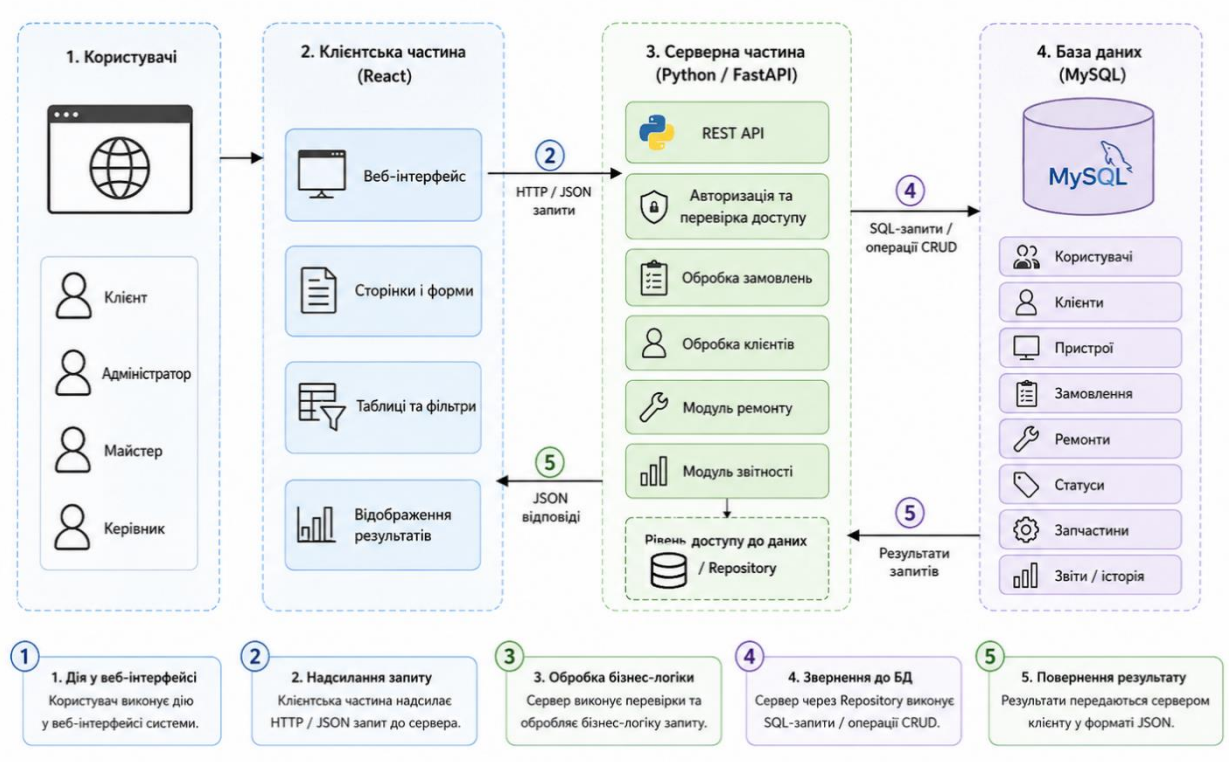

Рисунок 2.11 – Взаємодія програмних компонентів інформаційної системи

Перевагами обраної архітектури є розподіл відповідальності між компонентами, централізоване збереження даних, підтримка рольового доступу та можливість подальшого розширення системи.

Отже, клієнт-серверна багаторівнева архітектура відповідає функціональним вимогам розроблюваної системи. React використовується для

створення вебінтерфейсу, Python і FastAPI — для реалізації серверної логіки, а MySQL — для збереження даних.

2.5 Проєктування структури бази даних

База даних веб-орієнтованої інформаційної системи обліку замовлень призначена для централізованого збереження інформації про користувачів, клієнтів, пристрої, заявки, замовлення, ремонтні роботи, запчастини, статуси та оплати. Для реалізації рівня даних обрано реляційну систему керування базами даних MySQL [3].

Використання реляційної моделі є доцільним, оскільки основні сутності предметної області мають чітко визначені зв’язки. Реляційна модель передбачає подання даних у вигляді таблиць і встановлення зв’язків між ними за допомогою ключів [15]. Один клієнт може мати декілька пристроїв і замовлень, одному майстру може бути призначено декілька замовлень, а кожне замовлення може містити декілька виконаних робіт і використаних запчастин.

Під час проєктування бази даних було визначено такі основні сутності:

– ролі користувачів;

– користувачі системи;

– клієнти;

– пристрої;

– клієнтські заявки;

– замовлення;

– статуси замовлень;

– історія зміни статусів;

– результати діагностики;

– ремонтні роботи;

– запчастини;

– використані запчастини;

– оплати.

Центральною сутністю бази даних є замовлення. Воно пов’язане з клієнтом, пристроєм, відповідальним майстром і поточним статусом. Із замовленням також пов’язані результати діагностики, виконані ремонтні роботи, використані запчастини, історія зміни статусів і платежі.

Концептуальну модель бази даних наведено на рисунку 2.12.

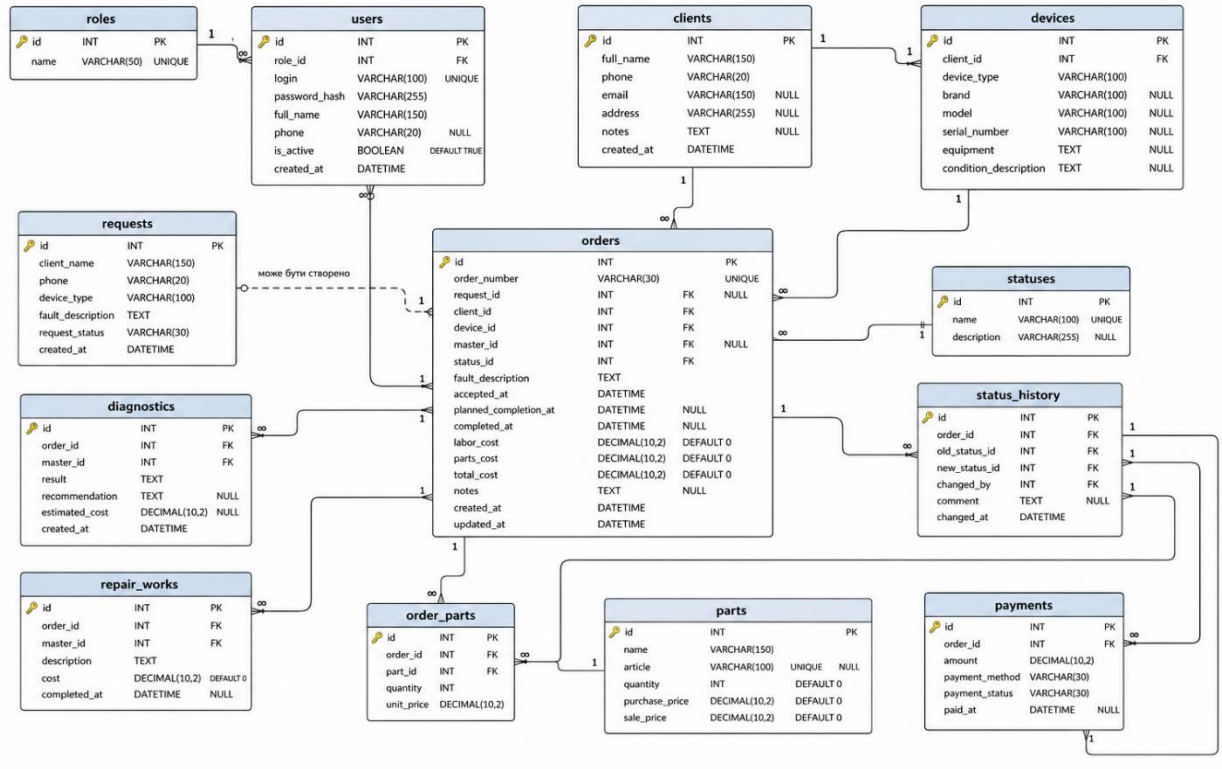

Рисунок 2.12 – Концептуальна модель бази даних інформаційної системи обліку замовлень

На рисунку 2.12 відображено основні таблиці бази даних і зв’язки між ними. Таблиця roles пов’язана з таблицею users відношенням «один до багатьох», оскільки одна роль може бути призначена декільком користувачам.

Таблиця clients пов’язана з таблицями devices та orders. Один клієнт може мати декілька пристроїв і декілька замовлень. Кожен пристрій належить одному клієнту, але може бути пов’язаний із кількома замовленнями в разі повторного звернення.

Таблиця orders є центральною таблицею бази даних. Вона містить зовнішні ключі для зв’язку з клієнтом, пристроєм, майстром і поточним статусом. З таблицею orders також пов’язані таблиці diagnostics, repair_works, status_history, order_parts і payments.

Зв’язок між замовленнями та запчастинами має тип «багато до багатьох». Одна запчастина може використовуватися в декількох замовленнях, а одне замовлення може містити декілька запчастин. Для реалізації цього зв’язку використовується проміжна таблиця order_parts.

Таблиця ролей користувачів roles призначена для збереження переліку ролей внутрішніх користувачів системи. У ній передбачено ролі адміністратора, майстра та керівника. Структуру таблиці roles наведено в таблиці 2.5.

Таблиця 2.5 – Структура таблиці roles

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Унікальний ідентифікатор ролі
name	VARCHAR(50)	NOT NULL, UNIQUE	Назва ролі

Таблиця користувачів users містить облікові записи внутрішніх користувачів системи. У ній зберігаються дані для авторизації, ПІБ користувача, його роль і стан облікового запису. Пароль користувача зберігається у вигляді хеш-значення. Структуру таблиці users наведено в таблиці 2.6.

Таблиця клієнтів clients призначена для збереження контактних даних клієнтів ремонтної майстерні. Вона дозволяє вести клієнтську базу та пов’язувати одного клієнта з декількома пристроями і замовленнями. Структуру таблиці clients наведено в таблиці 2.7.

Таблиця 2.6 – Структура таблиці users

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Ідентифікатор користувача
role_id	INT	FK, NOT NULL	Посилання на роль користувача
login	VARCHAR(100)	NOT NULL, UNIQUE	Логін для входу в систему
password_hash	VARCHAR(255)	NOT NULL	Хеш пароля
full_name	VARCHAR(150)	NOT NULL	ПІБ користувача
phone	VARCHAR(20)	NULL	Контактний номер телефону
is_active	BOOLEAN	NOT NULL, DEFAULT TRUE	Ознака активності облікового запису
created_at	DATETIME	NOT NULL	Дата створення облікового запису

Таблиця 2.7 – Структура таблиці clients

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Ідентифікатор клієнта
full_name	VARCHAR(150)	NOT NULL	ПІБ клієнта
phone	VARCHAR(20)	NOT NULL	Контактний номер телефону
email	VARCHAR(150)	NULL	Адреса електронної пошти
address	VARCHAR(255)	NULL	Адреса клієнта
notes	TEXT	NULL	Додаткові примітки
created_at	DATETIME	NOT NULL	Дата реєстрації клієнта

Таблиця пристроїв devices містить дані про пристрої, які передаються до майстерні. Винесення інформації про пристрої в окрему таблицю дозволяє зберігати історію повторних звернень із тим самим пристроєм. Структуру таблиці devices наведено в таблиці 2.8.

Таблиця клієнтських заявок requests призначена для збереження первинних заявок, поданих клієнтами через вебформу. Після перевірки заявки адміністратор може створити на її основі повноцінне замовлення. Структуру таблиці requests наведено в таблиці 2.9.

Таблиця 2.8 – Структура таблиці devices

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Ідентифікатор пристрою
client_id	INT	FK, NOT NULL	Посилання на власника пристрою
device_type	VARCHAR(100)	NOT NULL	Тип пристрою
brand	VARCHAR(100)	NULL	Виробник пристрою
model	VARCHAR(100)	NULL	Модель пристрою
serial_number	VARCHAR(100)	NULL	Серійний номер
equipment	TEXT	NULL	Комплектація під час приймання
condition_description	TEXT	NULL	Опис зовнішнього стану

Таблиця статусів statuses є довідником можливих станів замовлення. Вона містить такі значення, як «Нове замовлення», «Прийнято», «На діагностиці», «В роботі», «Очікує запчастини», «Виконано», «Видано клієнту» та «Скасовано». Структуру таблиці statuses наведено в таблиці 2.10.

Таблиця 2.9 – Структура таблиці requests

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Ідентифікатор заявки
client_name	VARCHAR(150)	NOT NULL	Ім’я заявника
phone	VARCHAR(20)	NOT NULL	Контактний номер телефону
device_type	VARCHAR(100)	NOT NULL	Тип пристрою
fault_description	TEXT	NOT NULL	Опис несправності
request_status	VARCHAR(30)	NOT NULL	Стан опрацювання заявки
created_at	DATETIME	NOT NULL	Дата та час подання заявки

Таблиця замовлень orders є центральною таблицею бази даних. Вона містить основну інформацію про замовлення, клієнта, пристрій, майстра, статус, строки виконання та вартість ремонту. Структуру таблиці orders наведено в таблиці 2.11.

Таблиця 2.10 – Структура таблиці statuses

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Ідентифікатор статусу
name	VARCHAR(100)	NOT NULL, UNIQUE	Назва статусу
description	VARCHAR(255)	NULL	Опис призначення статусу

Таблиця 2.11 – Структура таблиці orders

Поле	Тип даних	Обмеження	Призначення
1	2	3	4
id	INT	PK, AUTO_INCREMENT	Ідентифікатор замовлення
order_number	VARCHAR(30)	NOT NULL, UNIQUE	Публічний номер замовлення
request_id	INT	FK, NULL	Посилання на клієнтську заявку
client_id	INT	FK, NOT NULL	Посилання на клієнта
device_id	INT	FK, NOT NULL	Посилання на пристрій
master_id	INT	FK, NULL	Призначений майстер
status_id	INT	FK, NOT NULL	Поточний статус
fault_description	TEXT	NOT NULL	Опис несправності
accepted_at	DATETIME	NOT NULL	Дата та час приймання
planned_completion_at	DATETIME	NULL	Запланована дата завершення
completed_at	DATETIME	NULL	Фактична дата завершення
labor_cost	DECIMAL(10,2)	NOT NULL, DEFAULT 0	Вартість ремонтних робіт
parts_cost	DECIMAL(10,2)	NOT NULL, DEFAULT 0	Вартість використаних запчастин
total_cost	DECIMAL(10,2)	NOT NULL, DEFAULT 0	Загальна вартість замовлення
notes	TEXT	NULL	Додаткові примітки
created_at	DATETIME	NOT NULL	Дата створення запису
updated_at	DATETIME	NOT NULL	Дата останнього оновлення

Таблиця історії статусів status_history призначена для збереження історії переходів замовлення між статусами. Вона дозволяє визначити попередній і новий статус, користувача, який виконав зміну, та час виконання операції. Структуру таблиці status_history наведено в таблиці 2.12.

Таблиця 2.12 – Структура таблиці status_history

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Ідентифікатор запису історії
order_id	INT	FK, NOT NULL	Посилання на замовлення
old_status_id	INT	FK, NULL	Попередній статус
new_status_id	INT	FK, NOT NULL	Новий статус
changed_by	INT	FK, NOT NULL	Користувач, який змінив статус
comment	TEXT	NULL	Коментар до зміни
changed_at	DATETIME	NOT NULL	Дата та час зміни статусу

Таблиця результатів діагностики diagnostics містить результати перевірки пристрою, опис установленої причини несправності, рекомендації майстра та орієнтовну вартість ремонту. Структуру таблиці diagnostics наведено в таблиці 2.13.

Таблиця 2.13 – Структура таблиці diagnostics

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Ідентифікатор результату діагностики
order_id	INT	FK, NOT NULL	Посилання на замовлення
master_id	INT	FK, NOT NULL	Майстер, який виконав діагностику
result	TEXT	NOT NULL	Результат діагностики
recommendation	TEXT	NULL	Рекомендований спосіб ремонту
estimated_cost	DECIMAL(10,2)	NULL	Орієнтовна вартість ремонту
created_at	DATETIME	NOT NULL	Дата виконання діагностики

Таблиця ремонтних робіт repair_works призначена для збереження переліку робіт, виконаних у межах конкретного замовлення. Для кожної роботи зазначаються виконавець, опис, вартість і дата завершення. Структуру таблиці repair_works наведено в таблиці 2.14.

Таблиця запчастин parts використовується як довідник запчастин. Вона містить назву, артикул, кількість на складі, закупівельну та відпускну ціну. Структуру таблиці parts наведено в таблиці 2.15.

Таблиця 2.14 – Структура таблиці repair_works

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Ідентифікатор ремонтної роботи
order_id	INT	FK, NOT NULL	Посилання на замовлення
master_id	INT	FK, NOT NULL	Виконавець роботи
description	TEXT	NOT NULL	Опис виконаної роботи
cost	DECIMAL(10,2)	NOT NULL, DEFAULT 0	Вартість роботи
completed_at	DATETIME	NULL	Дата завершення роботи

Таблиця 2.15 – Структура таблиці parts

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Ідентифікатор запчастини
name	VARCHAR(150)	NOT NULL	Назва запчастини
article	VARCHAR(100)	NULL, UNIQUE	Артикул або складський код
quantity	INT	NOT NULL, DEFAULT 0	Кількість на складі
purchase_price	DECIMAL(10,2)	NOT NULL, DEFAULT 0	Закупівельна ціна
sale_price	DECIMAL(10,2)	NOT NULL, DEFAULT 0	Вартість для клієнта

Таблиця використаних запчастин order_parts є проміжною таблицею між замовленнями та запчастинами. Вона містить інформацію про кількість

використаних запчастин і ціну одиниці на момент ремонту. Структуру таблиці order_parts наведено в таблиці 2.16.

Таблиця оплат payments призначена для збереження інформації про платежі за виконані ремонтні роботи. Вона містить суму, спосіб оплати, стан платежу та дату його здійснення. Структуру таблиці payments наведено в таблиці 2.17.

Таблиця 2.16 – Структура таблиці order_parts

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Ідентифікатор запису
order_id	INT	FK, NOT NULL	Посилання на замовлення
part_id	INT	FK, NOT NULL	Посилання на запчастину
quantity	INT	NOT NULL	Використана кількість
unit_price	DECIMAL(10,2)	NOT NULL	Ціна одиниці на момент ремонту

Таблиця 2.17 – Структура таблиці payments

Поле	Тип даних	Обмеження	Призначення
id	INT	PK, AUTO_INCREMENT	Ідентифікатор платежу
order_id	INT	FK, NOT NULL	Посилання на замовлення
amount	DECIMAL(10,2)	NOT NULL	Сума платежу
payment_method	VARCHAR(30)	NOT NULL	Спосіб оплати
payment_status	VARCHAR(30)	NOT NULL	Стан платежу
paid_at	DATETIME	NULL	Дата та час оплати

У наведених таблицях використано такі позначення: PK — первинний ключ, FK — зовнішній ключ, NOT NULL — обов’язкове поле, UNIQUE — унікальне значення, AUTO_INCREMENT — автоматичне формування числового ідентифікатора.

Тип INT використовується для ідентифікаторів і цілих значень, VARCHAR — для коротких текстових даних, TEXT — для розгорнутих описів, DATETIME — для збереження дати та часу. Грошові значення зберігаються за допомогою типу DECIMAL(10,2), що забезпечує точність

фінансових розрахунків. Зазначені типи даних відповідають засобам організації таблиць у MySQL [3].

Під час проєктування структури бази даних виконано логічне розділення інформації на окремі таблиці. Це дозволяє зменшити дублювання записів, забезпечити цілісність даних і відповідає принципам нормалізації реляційних баз даних [15].

Для підтримки цілісності даних передбачено використання первинних і зовнішніх ключів, унікальність номера замовлення та логіна користувача, обов’язкове заповнення ключових полів, а також контроль невід’ємності вартості й кількості запчастин.

Отже, спроєктована структура бази даних охоплює основні сутності предметної області та забезпечує збереження інформації, необхідної для реєстрації, супроводження й завершення замовлень ремонтної майстерні.

2.6 Проєктування користувацького інтерфейсу

Користувацький інтерфейс веб-орієнтованої інформаційної системи обліку замовлень повинен забезпечувати зручну взаємодію користувача із функціями системи, швидкий доступ до основних операцій і зрозуміле відображення інформації. Під час проєктування інтерфейсу було враховано, що система використовується різними категоріями користувачів: клієнтом, адміністратором, майстром і керівником. Саме тому структура інтерфейсу будується з урахуванням ролей і прав доступу [10].

Основними вимогами до користувацького інтерфейсу є:

– простота та зрозумілість навігації;

– логічне групування елементів керування;

– мінімальна кількість дій для виконання типових операцій;

– єдиний стиль оформлення сторінок;

– адаптація інтерфейсу до ролі користувача;

– відображення зрозумілих повідомлень про помилки та результати виконання дій.

Простота, зрозумілість і мінімізація зайвих дій є важливими принципами проєктування інтерфейсу, оскільки вони впливають на ефективність і зручність роботи користувача із системою [11].

У структурі інтерфейсу доцільно виділити дві основні частини: публічну та авторизовану. Публічна частина призначена для клієнта і містить сторінки подання заявки на ремонт та перегляду статусу замовлення. Авторизована частина призначена для внутрішніх користувачів системи — адміністратора, майстра та керівника.

Для відображення логіки переходів між основними сторінками системи доцільно використати UML-діаграму переходів між інтерфейсами, яку наведено на рисунку 2.13.

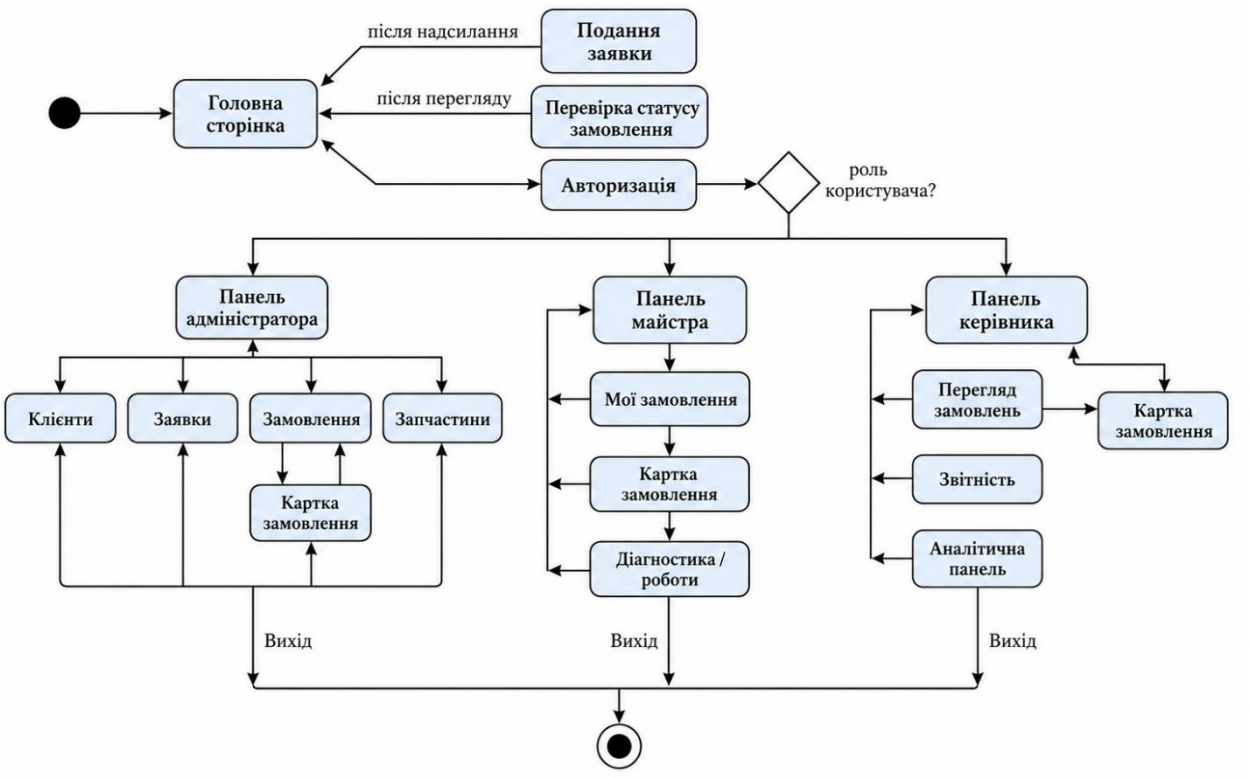

Рисунок 2.13 – UML-діаграма переходів між інтерфейсами користувача

На рисунку 2.13 доцільно показати основні інтерфейсні стани системи та переходи між ними. Із головної сторінки користувач може перейти до сторінки подання заявки, сторінки перевірки статусу замовлення або до сторінки авторизації. Після авторизації, залежно від ролі користувача, відкривається відповідний робочий інтерфейс: панель адміністратора, панель майстра або панель керівника. Із робочих панелей користувач переходить до сторінок списку замовлень, картки замовлення, роботи із клієнтами, запчастинами або звітністю.

Публічний інтерфейс клієнта повинен бути максимально простим. Його основне призначення полягає в поданні заявки на ремонт і перегляді поточного статусу замовлення. Шаблон публічного інтерфейсу наведено на рисунку 2.14.

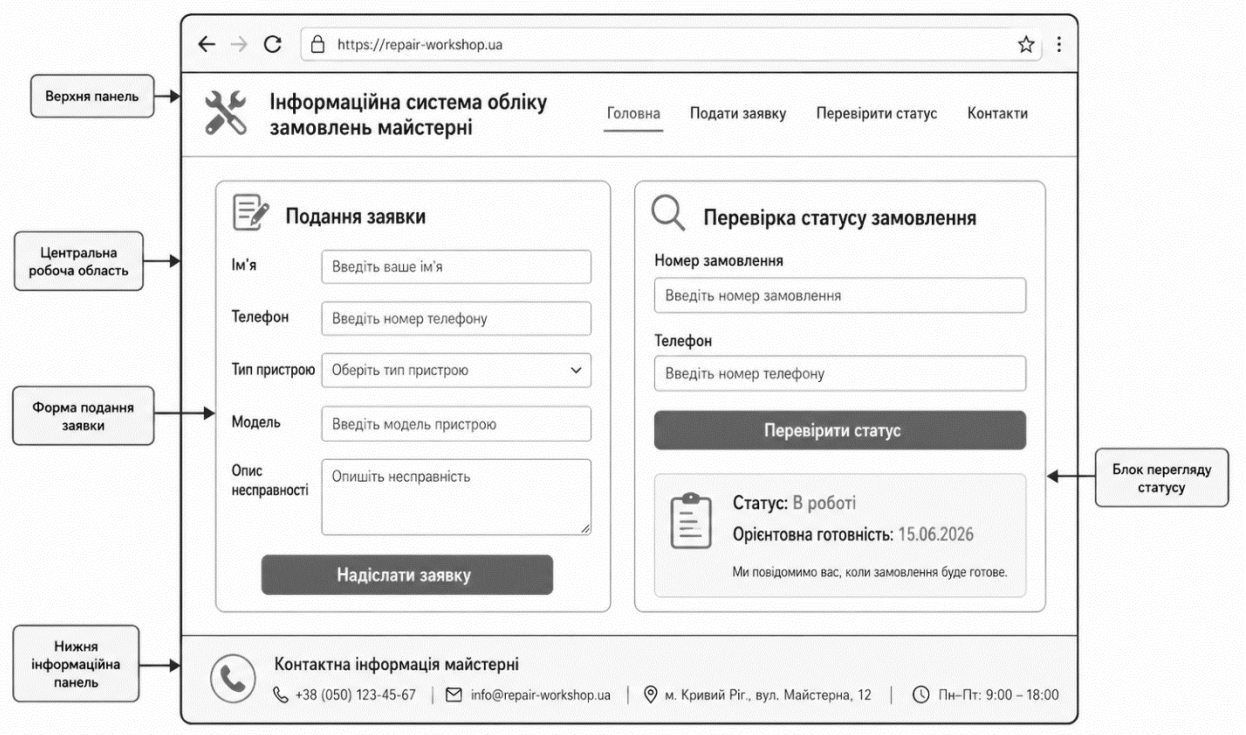

Рисунок 2.14 – Шаблон публічного інтерфейсу клієнта

У цьому шаблоні доцільно передбачити верхню панель із назвою системи, центральну робочу область і кнопки переходу до основних дій. Форма подання заявки повинна містити поля для введення імені клієнта, номера телефону, типу пристрою, моделі та опису несправності. Сторінка

перегляду статусу повинна містити поле введення номера замовлення та блок для відображення поточного стану ремонту.

Авторизований інтерфейс призначений для працівників майстерні. Його структура повинна бути єдиною для всіх внутрішніх користувачів, але наповнення залежить від ролі. Базовий шаблон такого інтерфейсу наведено на рисунку 2.15.

У шаблоні інтерфейсу внутрішнього користувача доцільно передбачити три основні області: верхню панель, бічне меню та центральну робочу область. Верхня панель містить назву системи та інформацію про авторизованого користувача. Бічне меню використовується для переходу між розділами. Центральна область призначена для відображення таблиць, форм, карток замовлень і звітної інформації.

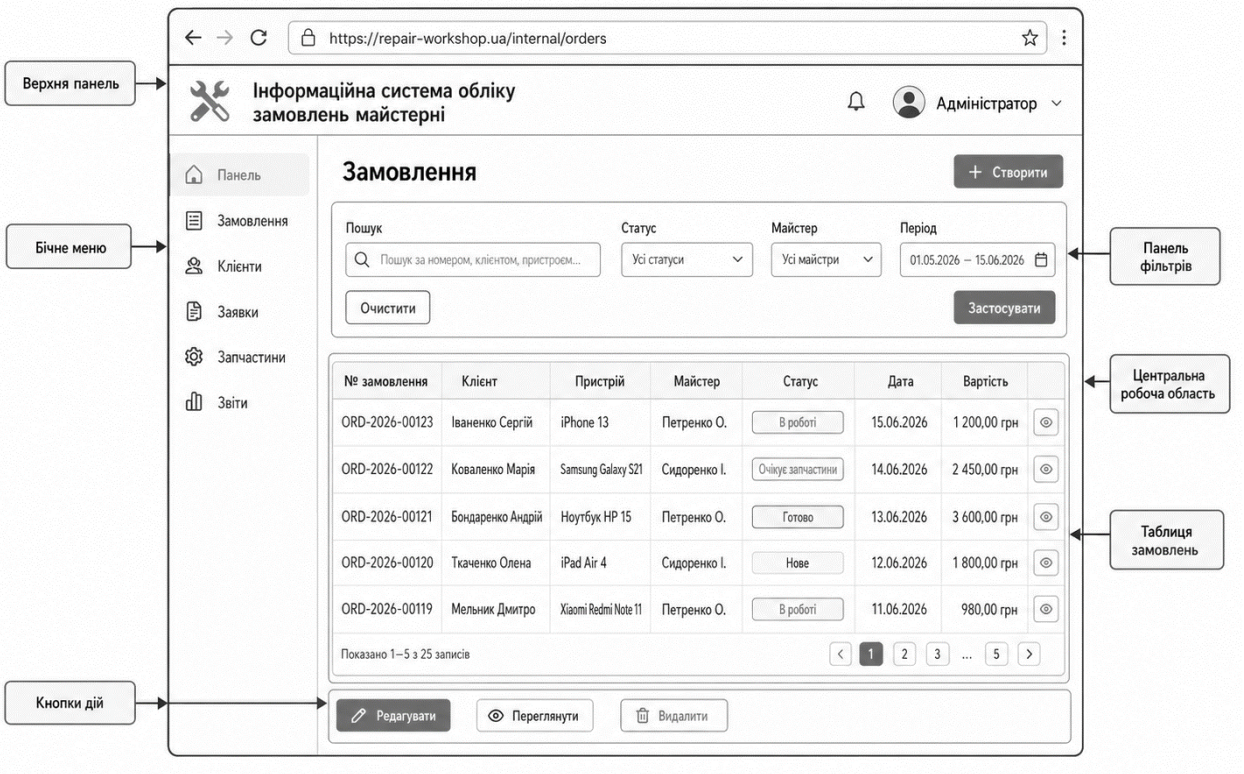

Рисунок 2.15 – Шаблон інтерфейсу внутрішнього користувача

Для адміністратора основними елементами інтерфейсу є сторінки роботи з клієнтами, заявками, замовленнями та запчастинами. Ключовою

сторінкою є список замовлень, у якому мають відображатися номер замовлення, клієнт, пристрій, майстер, статус, дата приймання та вартість. Також у цій сторінці повинні бути засоби пошуку та фільтрації.

Для майстра інтерфейс має бути спрощеним і орієнтованим насамперед на обробку призначених замовлень. Основними сторінками є список призначених замовлень, картка замовлення, форма внесення результатів діагностики, форма додавання виконаних робіт і форма додавання запчастин.

Для керівника доцільно передбачити інтерфейс у вигляді інформаційної панелі, на якій відображаються зведені показники роботи майстерні. Такий інтерфейс може містити кількість активних, завершених і скасованих замовлень, а також інформацію про завантаженість майстрів. Приклад шаблону сторінки перегляду робочої інформації наведено на рисунку 2.16.

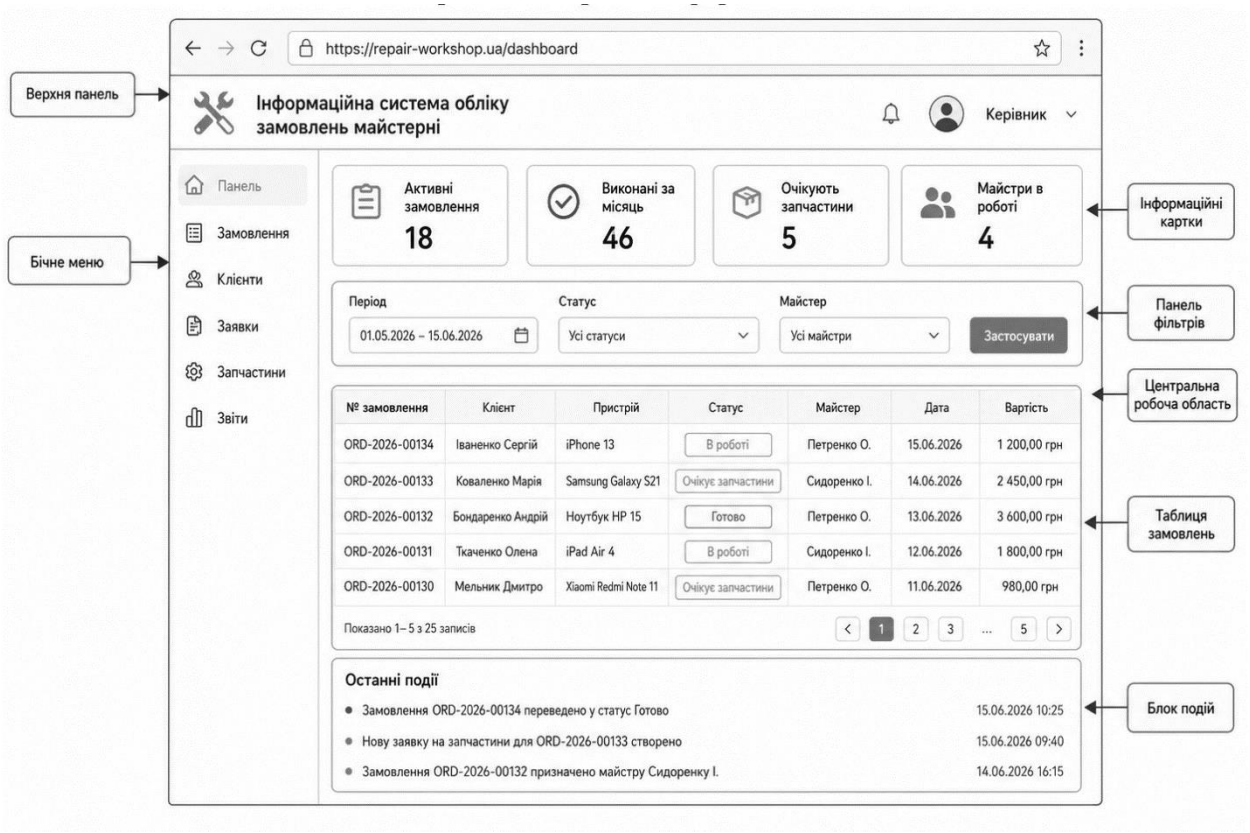

Рисунок 2.16 – Шаблон робочої сторінки інформаційної системи

На рисунку 2.16 доцільно відобразити таблицю замовлень або інформаційну панель із картками показників, блоком фільтрів і кнопками

основних дій. Така сторінка є типовою для більшості внутрішніх користувачів і може використовуватися як основа для реалізації конкретних екранів системи.

Основні елементи інтерфейсу та їх призначення наведено в таблиці 2.18.

Під час проєктування інтерфейсу також передбачено використання зрозумілих кнопок керування, таких як «Створити», «Зберегти», «Редагувати», «Додати роботу», «Додати запчастину», «Змінити статус» і «Скасувати». Усі форми повинні підтримувати перевірку обов’язкових полів, а система має повідомляти користувача про виявлені помилки.

Таблиця 2.18 – Основні елементи користувацького інтерфейсу

Елемент інтерфейсу	Призначення
Верхня панель	Відображення назви системи та даних користувача
Бічне меню	Навігація між основними розділами
Центральна робоча область	Відображення форм, таблиць, карток і звітів
Форма введення	Створення або редагування даних
Таблиця даних	Перегляд списків клієнтів, заявок і замовлень
Картка замовлення	Перегляд повної інформації про замовлення
Панель фільтрів	Пошук і відбір потрібних записів
Інформаційні повідомлення	Повідомлення про помилки та успішні дії

Отже, спроєктований користувацький інтерфейс базується на розподілі доступу за ролями, використанні єдиних шаблонів сторінок і логічній схемі переходів між інтерфейсами. Запропонований підхід створює основу для подальшої реалізації клієнтської частини системи засобами React.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Обґрунтування вибору засобів розробки

Для реалізації веб-орієнтованої інформаційної системи обліку замовлень ремонтної майстерні було обрано засоби, які відповідають клієнт-серверній архітектурі, забезпечують зручну роботу з даними та дозволяють розділити програмне забезпечення на незалежні компоненти.

До основних засобів розробки належать:

– бібліотека React для реалізації клієнтської частини [1];

– мова програмування Python і фреймворк FastAPI для серверної частини [2; 16];

– система керування базами даних MySQL [3];

– REST API для обміну даними між клієнтською та серверною частинами [17];

– HTML, CSS і JavaScript для побудови вебінтерфейсу [18];

– середовище Visual Studio Code для написання програмного коду [20];

– система контролю версій Git [21].

Клієнтську частину системи вирішено реалізувати з використанням React. Ця бібліотека дозволяє будувати користувацький інтерфейс із незалежних компонентів, які можна повторно використовувати на різних сторінках [1].

Компонентний підхід є зручним для розроблюваної системи, оскільки інтерфейс містить багато повторюваних елементів: форми введення, таблиці замовлень, панелі фільтрації, картки показників, кнопки та повідомлення.

До переваг React для цієї роботи належать:

– можливість створення односторінкового вебзастосунку;

– повторне використання компонентів;

– динамічне оновлення даних без повного перезавантаження сторінки;

– зручна реалізація форм, таблиць і фільтрів;

– можливість розмежування інтерфейсу відповідно до ролі користувача;

– підтримка маршрутизації між сторінками.

За допомогою React реалізуються публічний інтерфейс клієнта, панель адміністратора, робоче місце майстра та інформаційна панель керівника.

Для реалізації серверної частини обрано мову програмування Python. Вона має зрозумілий синтаксис, значну кількість бібліотек і добре підходить для створення вебзастосунків та роботи з базами даних [2].

Як серверний фреймворк обрано FastAPI. Він використовується для створення REST API, приймання HTTP-запитів, перевірки отриманих даних і формування відповідей клієнтській частині [16].

FastAPI забезпечує:

– опис маршрутів програмного інтерфейсу;

– обробку методів GET, POST, PATCH і DELETE;

– перевірку типів і структури вхідних даних;

– підтримку асинхронної обробки запитів;

– автоматичне формування документації API;

– зручну реалізацію авторизації та перевірки прав доступу.

У розроблюваній системі серверна частина відповідає за авторизацію користувачів, обробку заявок, роботу з клієнтами та замовленнями, зміну статусів, облік ремонтних робіт, запчастин і формування звітності.

Для збереження даних обрано реляційну систему керування базами даних MySQL. Вона дозволяє зберігати інформацію в окремих пов’язаних таблицях і підтримує використання первинних та зовнішніх ключів [3].

MySQL є доцільною для розроблюваної системи, оскільки дані мають чітку структуру. Замовлення пов’язуються з клієнтами, пристроями, майстрами, статусами, ремонтними роботами та запчастинами.

Основними перевагами MySQL є:

– підтримка реляційної моделі даних;

– використання мови SQL;

– забезпечення цілісності зв’язків між таблицями;

– підтримка індексів і транзакцій;

– можливість роботи з Python;

– достатня продуктивність для інформаційної системи невеликої майстерні.

Для доступу серверної частини до бази даних може використовуватися ORM-бібліотека SQLAlchemy або відповідний драйвер MySQL. Це дозволяє виконувати операції з таблицями засобами Python і зменшує кількість SQL-коду в програмних модулях [19].

Обмін даними між React і FastAPI здійснюється через REST API. Клієнтська частина формує HTTP-запити, а сервер повертає результати у форматі JSON [17].

Для виконання операцій використовуються такі методи:

– GET — отримання даних;

– POST — створення нового запису;

– PATCH — часткове оновлення запису;

– DELETE — видалення запису.

Використання зазначених методів відповідає принципам роботи HTTP-запитів у вебзастосунках [13]. Такий спосіб взаємодії дозволяє незалежно розробляти клієнтську та серверну частини, а також спрощує тестування програмного інтерфейсу.

Для написання та редагування програмного коду обрано Visual Studio Code. Середовище підтримує роботу з JavaScript, React, Python, SQL і Git, а також дозволяє встановлювати розширення для перевірки та форматування коду [20].

Для контролю змін у вихідному коді використовується Git. Його застосування дозволяє зберігати історію розробки, повертатися до попередніх версій і працювати з окремими гілками програмного проєкту [21].

Перелік основних засобів розробки наведено в таблиці 3.1.

Обраний набір засобів відповідає розробленій архітектурі системи. React забезпечує побудову інтерактивного інтерфейсу, Python і FastAPI реалізують

серверну логіку та програмний інтерфейс, а MySQL використовується для централізованого збереження даних.

Таблиця 3.1 – Засоби розробки інформаційної системи

Засіб	Призначення
React	Реалізація клієнтського веб-інтерфейсу
JavaScript	Програмування клієнтської логіки
HTML, CSS	Побудова структури та оформлення сторінок
Python	Реалізація серверної логіки
FastAPI	Створення REST API
MySQL	Зберігання даних системи
SQLAlchemy	Взаємодія Python із базою даних
Visual Studio Code	Написання та налагодження програмного коду
Git	Контроль версій програмного проєкту

Отже, використання React, Python, FastAPI та MySQL дозволяє реалізувати функціонально завершену веб-орієнтовану інформаційну систему, забезпечити розподіл відповідальності між її компонентами та створити можливість подальшого розширення програмного забезпечення.

3.2 Реалізація серверної частини системи засобами Python

Серверна частина веб-орієнтованої інформаційної системи реалізована мовою Python із використанням фреймворку FastAPI. Вона забезпечує обробку запитів клієнтської частини, перевірку даних, виконання бізнес-логіки, авторизацію користувачів і взаємодію з базою даних MySQL.

Серверний застосунок побудовано за модульним принципом. Програмний код поділено на окремі компоненти, кожен з яких відповідає за

конкретну групу функцій. Такий підхід спрощує тестування, супроводження та подальше розширення системи.

Орієнтовну структуру серверної частини наведено на рисунку 3.1.

[Тут розмістити схему структури серверної частини]

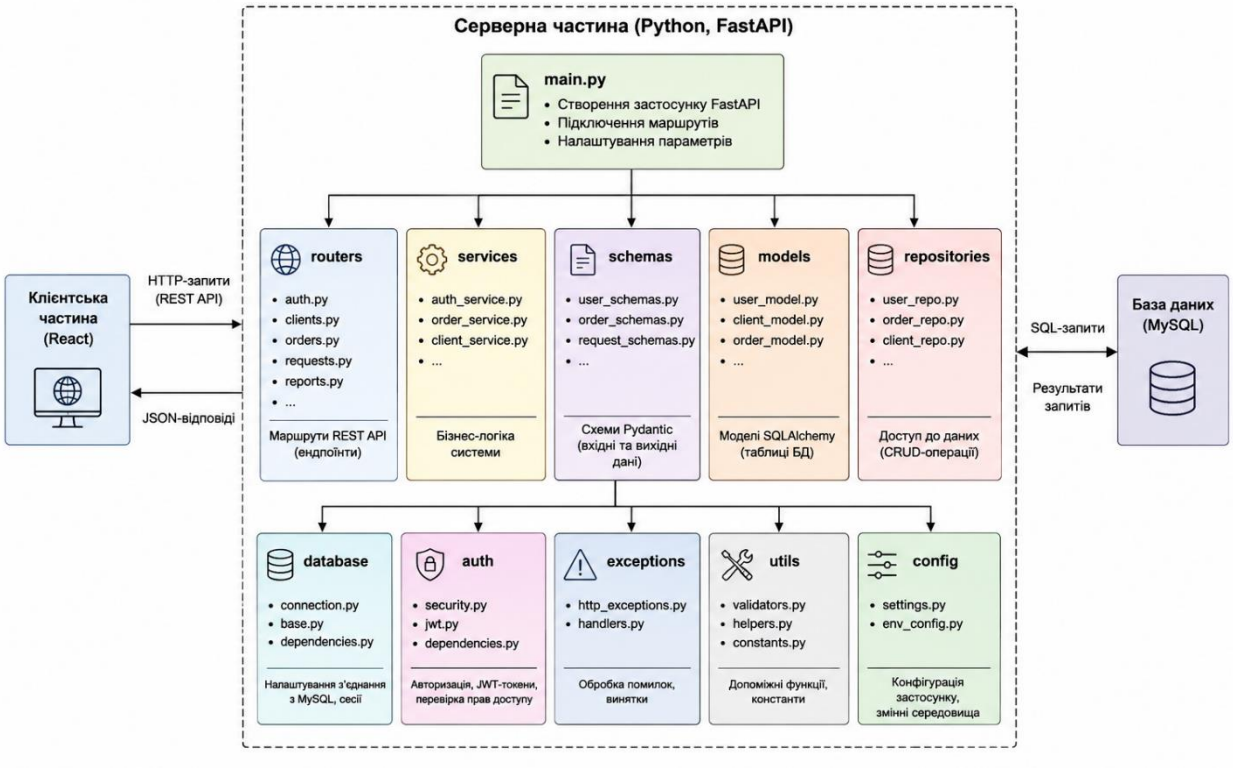

Рисунок 3.1 – Структура серверної частини інформаційної системи

У структурі серверного застосунку виділено такі основні модулі:

– main — запуск застосунку та підключення маршрутів;

– routers — маршрути REST API;

– models — моделі таблиць бази даних;

– schemas — структури вхідних і вихідних даних;

– services — бізнес-логіка системи;

– repositories — операції доступу до бази даних;

– database — налаштування з’єднання з MySQL;

– auth — авторизація, формування токенів і перевірка прав;

– exceptions — обробка програмних помилок.

Файл main.py є початковою точкою серверного застосунку. У ньому створюється об’єкт FastAPI, підключаються маршрути та налаштовуються параметри роботи сервера.

Фрагмент створення серверного застосунку має такий вигляд:

```
from fastapi import FastAPI

from app.routers import auth, clients, orders, requests, reports

app = FastAPI(
    title="Workshop Order Management API",
    version="1.0.0"
)

app.include_router(auth.router, prefix="/api/auth", tags=["Authorization"])
app.include_router(clients.router, prefix="/api/clients", tags=["Clients"])
app.include_router(orders.router, prefix="/api/orders", tags=["Orders"])
app.include_router(requests.router, prefix="/api/requests", tags=["Requests"])
app.include_router(reports.router, prefix="/api/reports", tags=["Reports"])
```

Маршрути серверної частини згруповано відповідно до функціональних модулів системи. Кожен маршрут визначає HTTP-метод, адресу запиту, вхідні параметри та формат відповіді.

Наприклад, маршрут отримання списку замовлень обробляє GET-запит, перевіряє права користувача та повертає дані у форматі JSON.

```
@router.get("/")
def get_orders(
    status_id: int | None = None,
    master_id: int | None = None,
    current_user: User = Depends(get_current_user),
    session: Session = Depends(get_session)
):
    return order_service.get_orders(
        session=session,
        status_id=status_id,
        master_id=master_id,
        current_user=current_user
    )
```

Для опису структури вхідних і вихідних даних використовуються схеми Pydantic. Вони забезпечують автоматичну перевірку типів і обов’язкових полів.

Схема створення клієнтської заявки може мати такий вигляд:

```
from pydantic import BaseModel, Field

class RequestCreate(BaseModel):
    client_name: str = Field(min_length=2, max_length=150)
    phone: str = Field(min_length=10, max_length=20)
    device_type: str = Field(min_length=2, max_length=100)
    model: str | None = Field(default=None, max_length=100)
    fault_description: str = Field(min_length=5)
```

Якщо отримані дані не відповідають визначеній схемі, FastAPI автоматично формує повідомлення про помилку. Це зменшує кількість додаткового коду для перевірки введених значень.

Для взаємодії з базою даних використовується SQLAlchemy. Кожна таблиця бази даних представлена окремою моделлю Python.

Приклад моделі замовлення:

```
from sqlalchemy import DateTime, ForeignKey, Numeric, String, Text
from sqlalchemy.orm import Mapped, mapped_column

class Order(Base):
    __tablename__ = "orders"

    id: Mapped[int] = mapped_column(primary_key=True)
    order_number: Mapped[str] = mapped_column(
        String(30),
        unique=True,
        nullable=False
    )
    client_id: Mapped[int] = mapped_column(
        ForeignKey("clients.id"),
        nullable=False
    )
    device_id: Mapped[int] = mapped_column(
        ForeignKey("devices.id"),
        nullable=False
    )
    master_id: Mapped[int | None] = mapped_column(
```

```
        ForeignKey("users.id")
    )
    status_id: Mapped[int] = mapped_column(
        ForeignKey("statuses.id"),
        nullable=False
    )
    fault_description: Mapped[str] = mapped_column(
        Text,
        nullable=False
    )
    total_cost: Mapped[float] = mapped_column(
        Numeric(10, 2),
        default=0
    )
```

Операції з базою даних винесено до рівня репозиторіїв. Репозиторій виконує пошук, додавання, оновлення та видалення записів, не містячи при цьому бізнес-логіки.

Сервісний рівень виконує перевірку правил предметної області. Наприклад, під час створення замовлення перевіряється наявність клієнта та пристрою, формується унікальний номер, установлюється початковий статус і створюється перший запис в історії статусів.

Послідовність створення замовлення серверною частиною наведено на рисунку 3.2.

Унікальний номер замовлення формується автоматично. Він може містити префікс, поточний рік і порядковий номер, наприклад ORD-2026-00125. Такий формат є зручним для пошуку та використання клієнтом під час перевірки статусу.

Авторизація внутрішніх користувачів реалізується за допомогою токенів доступу. Після введення логіна й пароля сервер шукає користувача в базі даних і перевіряє хеш пароля. У разі успішної перевірки формується токен, який надалі передається клієнтською частиною разом із захищеними запитами.

Права доступу визначаються роллю користувача. Наприклад, створення замовлення доступне адміністратору, внесення результатів діагностики — майстру, а отримання зведеної звітності — керівнику.

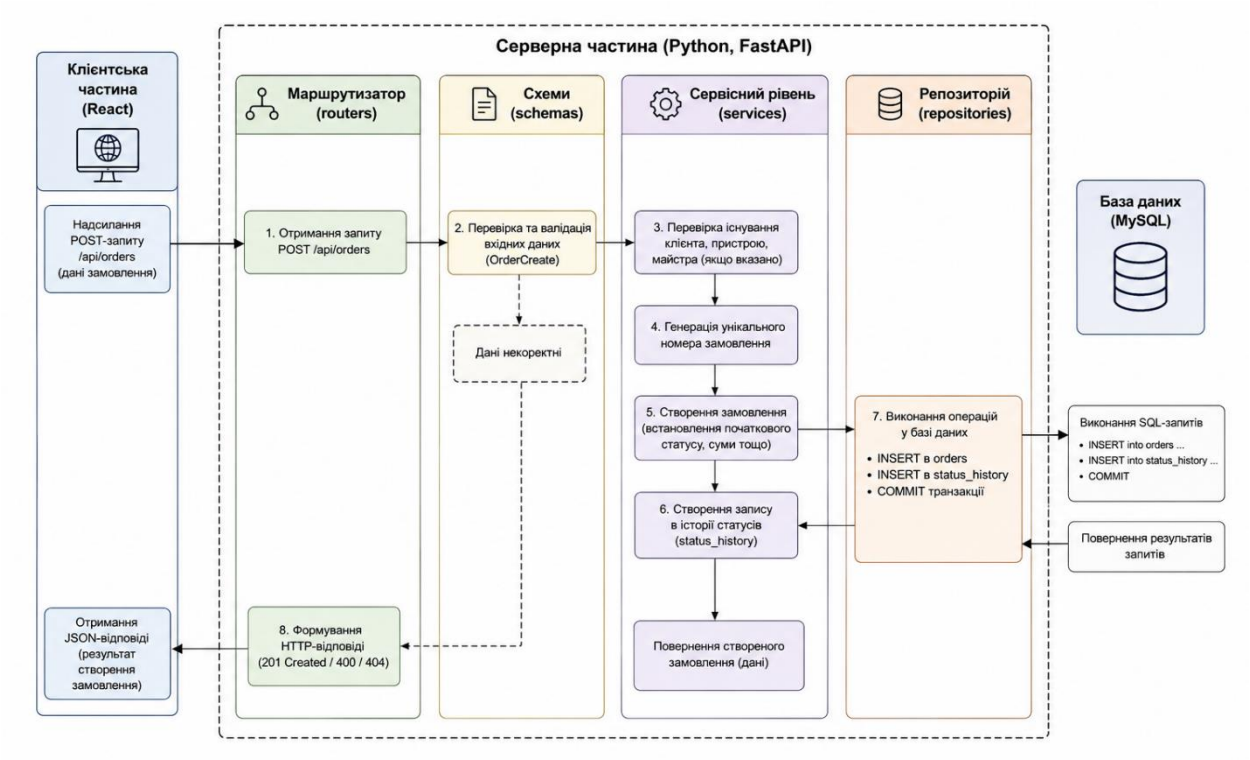

Рисунок 3.2 – Обробка серверною частиною запиту створення замовлення

Перевірка ролі може виконуватися за допомогою залежності FastAPI:

```
def require_roles(*allowed_roles: str):
    def role_checker(
        current_user: User = Depends(get_current_user)
    ) -> User:
        if current_user.role.name not in allowed_roles:
            raise HTTPException(
                status_code=403,
                detail="Недостатньо прав доступу"
            )
        return current_user

    return role_checker
```

Для зміни статусу замовлення сервер перевіряє існування замовлення, допустимість нового статусу та права користувача. Після оновлення поля status_id у таблиці orders створюється запис у таблиці status_history.

Це дозволяє зберігати не лише поточний стан замовлення, але й повну історію його проходження через етапи ремонту.

Для клієнта реалізується відкритий маршрут перегляду статусу замовлення. Для отримання інформації клієнт вводить номер замовлення та номер телефону. Сервер порівнює отримані дані з даними в базі та повертає лише обмежену інформацію: номер замовлення, поточний статус, орієнтовну дату завершення та повідомлення про готовність.

Серверна частина також забезпечує обробку помилок. Основними типами помилок є:

– некоректні вхідні дані;

– відсутність потрібного запису;

– неправильний логін або пароль;

– недостатні права доступу;

– недопустимий перехід між статусами;

– помилка взаємодії з базою даних.

У разі виникнення помилки сервер повертає HTTP-код і коротке повідомлення. Наприклад, код 400 використовується для некоректного запиту, 401 — для помилки авторизації, 403 — для заборони доступу, 404 — якщо запис не знайдено, а 500 — для внутрішньої помилки сервера.

Основні програмні модулі серверної частини наведено в таблиці 3.2.

Таблиця 3.2 – Модулі серверної частини інформаційної системи

Модуль	Призначення
auth	Авторизація, перевірка паролів і формування токенів
clients	Робота з даними клієнтів
devices	Облік пристроїв
requests	Обробка клієнтських заявок
orders	Створення, перегляд і редагування замовлень
diagnostics	Збереження результатів діагностики
repair_works	Облік виконаних робіт
parts	Робота із запчастинами
statuses	Зміна статусів та ведення історії
payments	Облік оплат
reports	Формування зведеної інформації

Отже, серверна частина, реалізована мовою Python із використанням FastAPI, забезпечує виконання основної бізнес-логіки системи, взаємодію з базою даних MySQL і розмежування доступу відповідно до ролей користувачів. Модульна структура програмного коду спрощує тестування, супроводження та подальше розширення інформаційної системи.

3.3 Реалізація клієнтської частини системи засобами React

Клієнтська частина веб-орієнтованої інформаційної системи реалізована з використанням бібліотеки React. Вона забезпечує відображення користувацького інтерфейсу, введення та попередню перевірку даних, навігацію між сторінками, а також обмін інформацією із серверною частиною через REST API [1; 17].

Під час реалізації клієнтської частини було використано компонентний підхід. Інтерфейс поділено на окремі повторно використовувані компоненти: форми, таблиці, панелі фільтрів, кнопки, інформаційні картки, повідомлення та елементи навігації. Така організація спрощує супроводження програмного коду та забезпечує однакове оформлення сторінок.

Клієнтський застосунок має модульну структуру. Орієнтовну структуру React-проєкту наведено на рисунку 3.3.

Початковим компонентом клієнтського застосунку є App. У ньому підключаються маршрути та визначається структура переходів між сторінками.

Для маршрутизації використовується бібліотека React Router. Вона дозволяє реалізувати переходи між сторінками без повного перезавантаження вебзастосунку.

Публічна частина містить сторінку подання заявки та сторінку перевірки статусу замовлення. Форма подання заявки реалізується як керований React-компонент, у якому значення полів зберігаються у стані компонента.

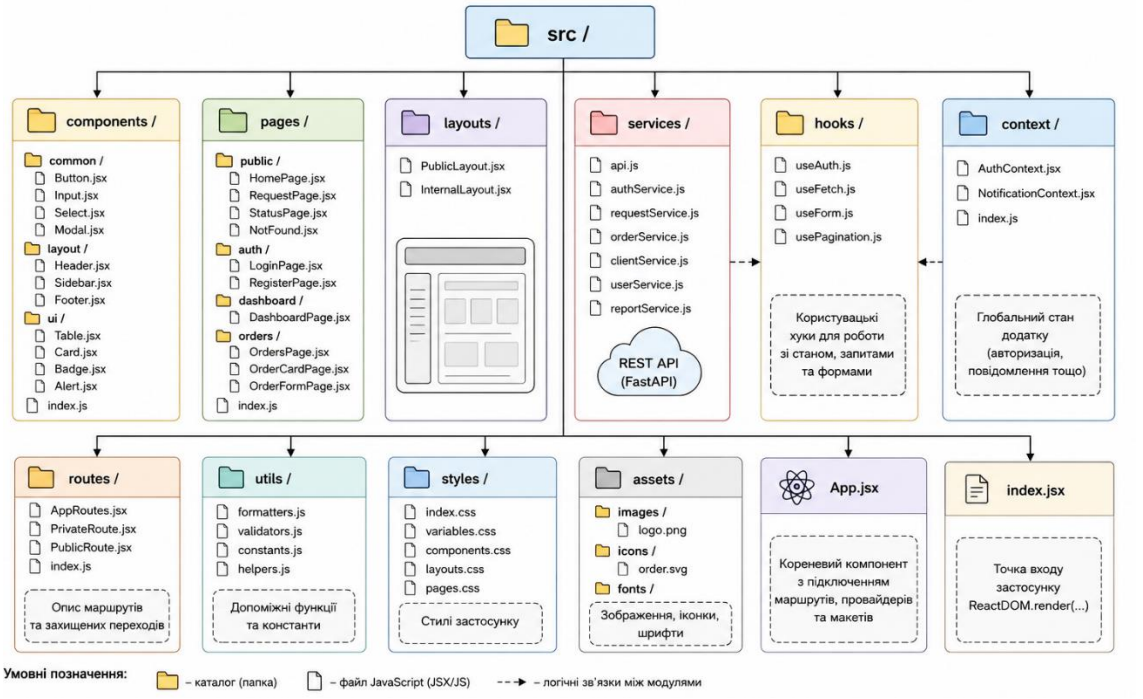

Рисунок 3.3 – Структура клієнтської частини інформаційної системи

Взаємодія із серверною частиною винесена до окремих сервісних модулів. Для надсилання HTTP-запитів може використовуватися бібліотека Axios:

```
import axios from "axios";

const api = axios.create({
  baseURL: "http://localhost:8000/api",
});

export const createRequest = async (requestData) => {
  const response = await api.post("/requests", requestData);
  return response.data;
};
```

Таке розділення дозволяє не розміщувати логіку HTTP-запитів безпосередньо в компонентах інтерфейсу.

Для авторизації внутрішніх користувачів реалізується форма входу. Після успішного введення логіна й пароля сервер повертає токен доступу та інформацію про роль користувача. Токен використовується для виконання наступних захищених запитів.

Стан авторизації доцільно зберігати за допомогою React Context. Це дозволяє отримувати дані про поточного користувача в різних компонентах без багаторазового передавання параметрів.

Для обмеження доступу до внутрішніх сторінок використовується компонент захищеного маршруту. Він перевіряє наявність токена та роль користувача.

```
import { Navigate } from "react-router-dom";
import { useAuth } from "../context/AuthContext";

function ProtectedRoute({ children, allowedRoles }) {
  const { user, token } = useAuth();

  if (!token) {
    return <Navigate to="/login" replace />;
  }

  if (allowedRoles && !allowedRoles.includes(user?.role)) {
    return <Navigate to="/" replace />;
  }

  return children;
}
```

Основною сторінкою внутрішнього інтерфейсу є список замовлень. Дані отримуються із серверної частини та відображаються у вигляді таблиці. Для зручності користувача передбачаються пошук, фільтрація за статусом і майстром, а також перехід до картки замовлення.

Картка замовлення реалізується як окрема сторінка. Вона містить дані клієнта та пристрою, опис несправності, поточний статус, результати діагностики, перелік виконаних робіт, використані запчастини та історію змін.

Склад доступних елементів залежить від ролі користувача. Адміністратор може редагувати основні дані замовлення та призначати майстра. Майстер може вносити результати діагностики, додавати виконані роботи й запчастини. Керівник має доступ переважно в режимі перегляду.

Для керівника реалізується інформаційна панель, що містить картки зі зведеними показниками: кількість активних, виконаних і скасованих

замовлень, замовлення, що очікують запчастини, та кількість майстрів у роботі.

Типову взаємодію клієнтської частини із серверною під час завантаження списку замовлень наведено на рисунку 3.4.

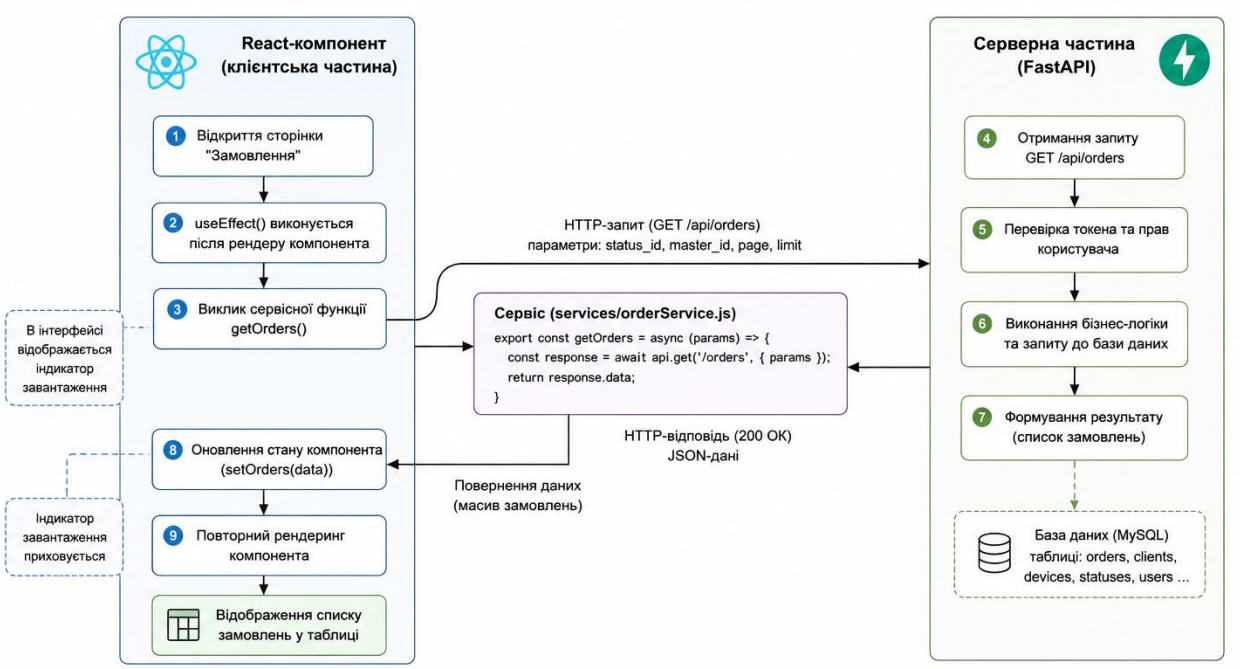

Рисунок 3.4 – Взаємодія клієнтської частини із серверною під час отримання даних

Після відкриття сторінки React-компонент виконує запит до REST API. Сервер повертає JSON-відповідь, після чого компонент оновлює свій стан і повторно відображає інтерфейс із отриманими даними.

Важливою частиною реалізації є обробка станів завантаження та помилок. Під час отримання даних користувачу відображається індикатор завантаження, а в разі помилки — відповідне повідомлення :

```
if (isLoading) {
  return <p>Завантаження даних...</p>;
}

if (error) {
  return <p>Помилка отримання даних</p>;
}
```

Для оформлення інтерфейсу використовуються CSS-стилі або бібліотека готових компонентів. Інтерфейс побудовано за єдиним шаблоном, який містить верхню панель, бічне меню та центральну робочу область.

Клієнтська частина, реалізована засобами React, забезпечує зручну взаємодію користувачів із функціями інформаційної системи. Компонентний підхід, маршрутизація, рольове розмежування доступу та окремі сервіси роботи з REST API створюють зрозумілу структуру клієнтського застосунку та спрощують його подальше розширення.

3.4 Реалізація бази даних MySQL

База даних інформаційної системи реалізована в середовищі MySQL відповідно до структури, спроєктованої у підрозділі 2.5. Вона забезпечує централізоване збереження даних про користувачів, клієнтів, пристрої, заявки, замовлення, результати діагностики, виконані роботи, запчастини, статуси та оплати [3].

Створення бази даних виконується за допомогою SQL-скриптів. Під час реалізації використовуються первинні й зовнішні ключі, обмеження цілісності, унікальні індекси та значення за замовчуванням. Використання ключів і обмежень дозволяє підтримувати цілісність реляційних даних і зменшити ризик появи некоректних записів [15].

Створення бази даних виконується такою командою:

```
CREATE DATABASE workshop_orders
CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;

USE workshop_orders;
```

Кодування utf8mb4 забезпечує коректне збереження українського тексту, спеціальних символів і даних, що вводяться користувачами.

Першими створюються довідникові таблиці, які не залежать від інших сутностей. Приклад створення таблиці ролей:

```
CREATE TABLE roles (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL UNIQUE
);
```

Після цього створюється таблиця користувачів системи:

Центральною таблицею бази даних є orders. Вона містить основні відомості про замовлення та зовнішні ключі до пов’язаних сутностей.

Для збереження історії зміни статусів створено таблицю status_history:

Для збереження результатів діагностики та виконаних робіт реалізовано таблиці diagnostics і repair_works.

ля обліку запчастин використано таблиці parts і order_parts. Таблиця parts зберігає загальні відомості про запчастини, а order_parts пов’язує їх із конкретними замовленнями.

Підсумкова вартість замовлення може визначатися як сума вартості ремонтних робіт і використаних запчастин:

```
UPDATE orders
SET total_cost = labor_cost + parts_cost
WHERE id = ?;
```

Для таблиць, за якими часто виконується пошук, створюються індекси:

```
CREATE INDEX idx_orders_status
ON orders(status_id);

CREATE INDEX idx_orders_master
ON orders(master_id);

CREATE INDEX idx_orders_client
ON orders(client_id);

CREATE INDEX idx_orders_accepted_at
ON orders(accepted_at);

CREATE INDEX idx_clients_phone
```

```
ON clients(phone);
```

Індекси прискорюють пошук замовлень за статусом, майстром, клієнтом і датою приймання.

Початкові значення ролей і статусів додаються до бази даних під час її налаштування.

Для перевірки роботи бази даних виконуються тестові операції додавання, вибірки, оновлення та видалення записів. Наприклад, список активних замовлень може бути отриманий таким запитом:

```
SELECT
    o.order_number,
    c.full_name AS client_name,
    d.device_type,
    d.model,
    s.name AS status_name,
    u.full_name AS master_name,
    o.total_cost
FROM orders o
JOIN clients c ON c.id = o.client_id
JOIN devices d ON d.id = o.device_id
JOIN statuses s ON s.id = o.status_id
LEFT JOIN users u ON u.id = o.master_id
WHERE s.name NOT IN ('Видано клієнту', 'Скасовано')
ORDER BY o.accepted_at DESC;
```

Результати такого запиту використовуються серверною частиною для формування списку активних замовлень у клієнтському інтерфейсі.

Отже, реалізована база даних MySQL відповідає спроєктованій логічній моделі, підтримує зв’язки між основними сутностями, забезпечує цілісність даних і дозволяє виконувати необхідні операції обліку замовлень ремонтної майстерні.

3.5 Організація взаємодії між компонентами системи

Взаємодія між компонентами веб-орієнтованої інформаційної системи організована відповідно до клієнт-серверної архітектури. Клієнтська частина, реалізована засобами React, відповідає за відображення інтерфейсу та формування запитів [1]. Серверна частина на основі Python і FastAPI виконує перевірку даних, бізнес-логіку та контроль прав доступу [16]. База даних MySQL забезпечує збереження й отримання інформації [3].

Основний обмін даними між клієнтською та серверною частинами здійснюється через REST API [17]. Запити передаються за протоколом HTTP, а дані — у форматі JSON [13; 22]. Для виконання операцій використовуються методи GET, POST, PATCH і DELETE.

Загальна послідовність взаємодії компонентів має такий вигляд:

– користувач виконує дію у вебінтерфейсі;

– React-компонент формує запит до REST API;

– FastAPI приймає запит і перевіряє його структуру;

– сервер визначає користувача та перевіряє його права;

– сервісний модуль виконує бізнес-логіку;

– репозиторій звертається до бази даних MySQL;

– база даних повертає результат виконання операції;

– сервер формує JSON-відповідь;

– клієнтська частина оновлює стан і відображає результат користувачу.

Схему взаємодії основних компонентів системи наведено на рисунку 3.5.

Під час відкриття сторінки замовлень React-компонент виконує GET-запит до маршруту /api/orders. Сервер перевіряє токен доступу, виконує запит до бази даних і повертає список замовлень. Після отримання відповіді клієнтська частина оновлює свій стан і відображає записи у таблиці.

Під час створення нового замовлення клієнтська частина надсилає POST-запит із даними клієнта, пристрою, описом несправності та

призначеним майстром. Сервер перевіряє коректність полів, формує унікальний номер замовлення, створює запис у таблиці orders і додає початковий запис до таблиці status_history.

Приклад структури запиту створення замовлення:

```
{
  "client_id": 12,
  "device_id": 8,
  "master_id": 4,
  "status_id": 1,
  "fault_description": "Пристрій не вмикається",
  "planned_completion_at": "2026-06-20T18:00:00"
}
```

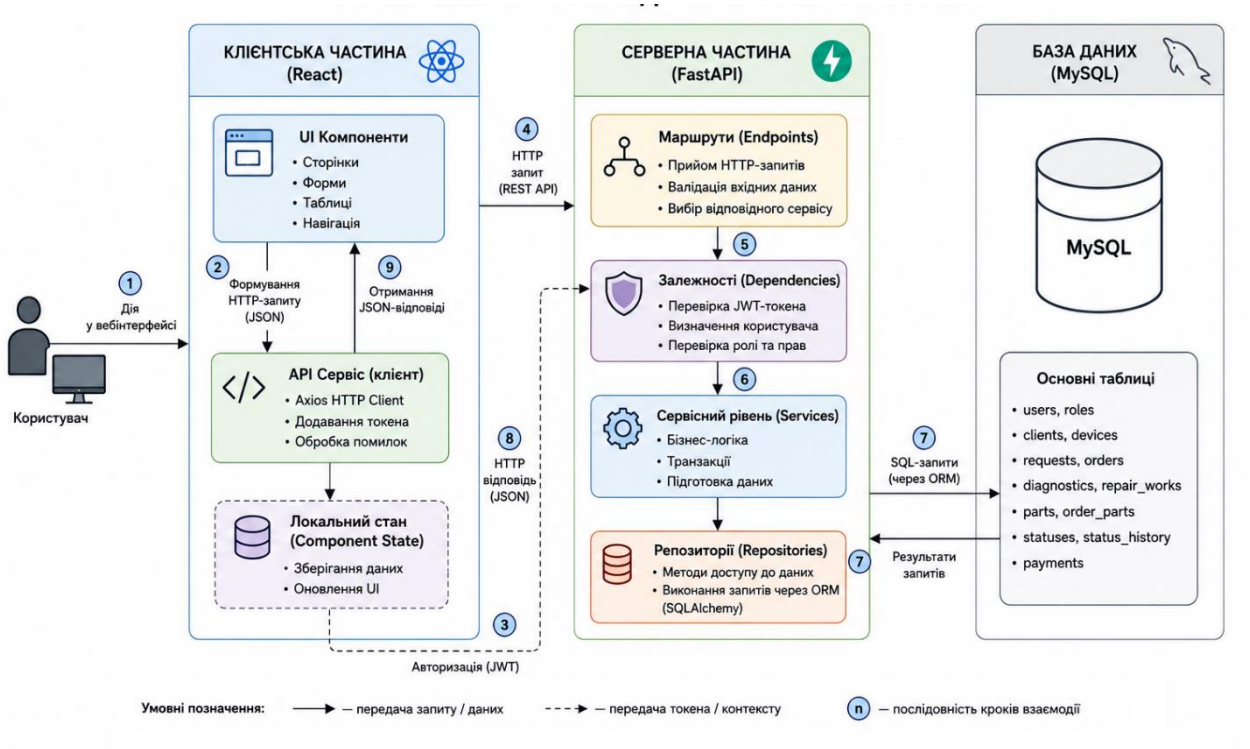

Рисунок 3.5 – Загальна схема взаємодії компонентів інформаційної системи

У разі успішного виконання сервер повертає код 201 Created і дані створеного замовлення:

```
{
  "id": 125,
  "order_number": "ORD-2026-00125",
  "status": "Нове замовлення"
}
```

Для захищених маршрутів клієнтська частина передає токен у заголовку Authorization: Authorization: Bearer <access_token>

FastAPI перевіряє токен, визначає користувача та його роль. Якщо користувач не має необхідних прав, сервер повертає код 403 Forbidden. Такий механізм застосовується до операцій створення замовлень, внесення результатів ремонту та перегляду звітності.

Під час зміни статусу замовлення клієнтська частина надсилає PATCH-запит. Сервер оновлює поле status_id у таблиці orders і створює запис у таблиці status_history. Завдяки цьому зберігається історія переходів замовлення між етапами ремонту.

Для обробки помилок використовується єдиний формат відповіді:

```
{
  "detail": "Замовлення не знайдено"
}
```

Клієнтська частина аналізує код відповіді та відображає відповідне повідомлення. Основні коди стану, що використовуються в системі, наведено в таблиці 3.4.

Таблиця 3.4 – Основні коди HTTP-відповідей системи

Код	Значення	Використання
200	OK	Успішне отримання або оновлення даних
201	Created	Успішне створення нового запису
400	Bad Request	Некоректні вхідні дані
401	Unauthorized	Відсутня або неправильна авторизація
403	Forbidden	Недостатньо прав доступу
404	Not Found	Запис не знайдено
422	Unprocessable Entity	Помилка перевірки структури даних
500	Internal Server Error	Внутрішня помилка сервера

Взаємодія з базою даних виконується через репозиторії та SQLAlchemy. Сервісні модулі не формують SQL-запити безпосередньо, а викликають

методи рівня доступу до даних. Це зменшує залежність бізнес-логіки від конкретної реалізації бази даних.

Для операцій, що складаються з декількох пов’язаних змін, використовуються транзакції. Наприклад, під час створення замовлення необхідно одночасно додати запис до таблиці orders і початковий запис до status_history. Якщо одна з операцій завершується помилкою, зміни скасовуються.

Типову послідовність обробки запиту наведено на рисунку 3.6.

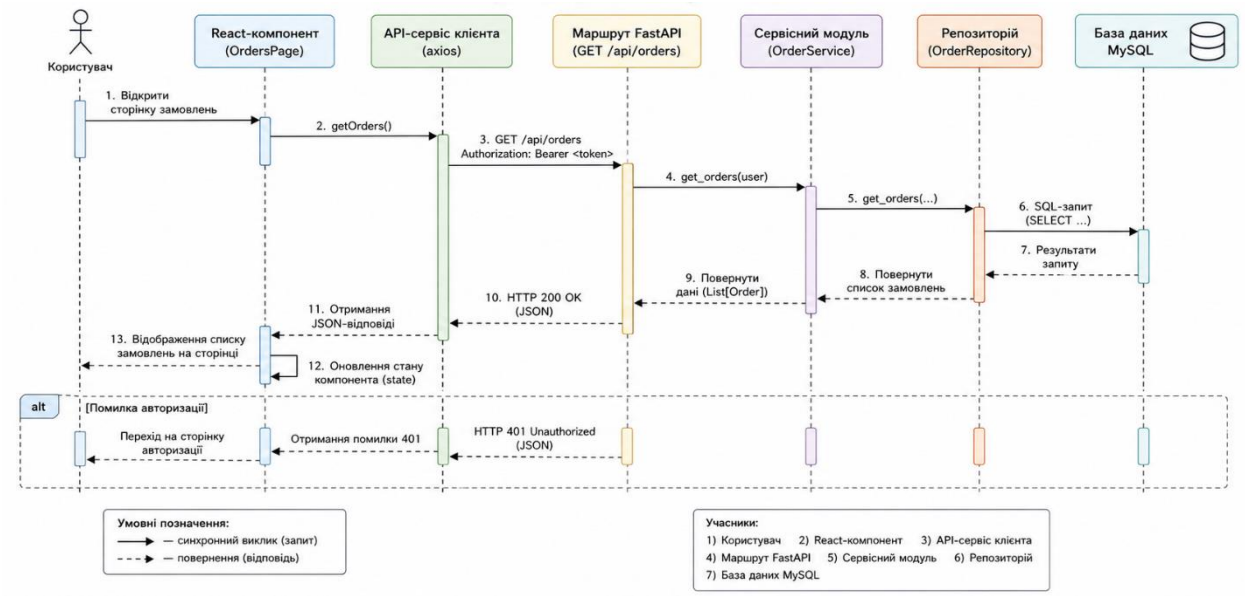

Рисунок 3.6 – UML-діаграма послідовності взаємодії компонентів системи

На рисунку 3.6 доцільно відобразити таких учасників: користувач, React-компонент, API-сервіс клієнтської частини, маршрут FastAPI, сервісний модуль, репозиторій і база даних MySQL. Діаграма повинна показувати передачу запиту, перевірку даних, виконання операції та повернення відповіді.

Отже, взаємодія між компонентами системи побудована на використанні REST API, формату JSON, рольового контролю доступу та багаторівневої обробки запитів. Така організація забезпечує розподіл відповідальності між клієнтською частиною, серверною логікою та базою даних.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ СИСТЕМИ

4.1 Методика тестування

Тестування веб-орієнтованої інформаційної системи обліку замовлень проводиться з метою перевірки правильності реалізації функціональних вимог, стабільності взаємодії між React, FastAPI та MySQL, а також коректності рольового доступу [1; 3; 10; 16].

Об’єктами тестування є публічний інтерфейс клієнта, авторизація внутрішніх користувачів, модулі клієнтів, пристроїв, заявок, замовлень, ремонтних робіт, статусів, запчастин, звітності, REST API та база даних [17].

Основним методом є функціональне тестування за принципом «чорної скриньки». Для кожної функції задаються вхідні дані, послідовність дій, очікуваний і фактичний результати. Такий підхід дозволяє перевірити поведінку системи без аналізу її внутрішнього програмного коду [23]. Додатково застосовується інтеграційне тестування для перевірки обміну даними між клієнтською частиною, сервером і базою даних [24].

Методика тестування включає такі етапи:

– підготовку тестового середовища;

– формування тестових сценаріїв;

– введення тестових даних;

– виконання операцій;

– порівняння отриманого результату з очікуваним;

– фіксацію помилок;

– повторну перевірку після виправлення.

Тестове середовище складається з клієнтського застосунку React, сервера FastAPI та бази даних MySQL, запущених локально. Перевірка інтерфейсу виконується у веббраузері, а REST API — за допомогою вбудованої документації FastAPI або засобу надсилання HTTP-запитів [16; 17].

Функціональне тестування охоплює подання заявки, створення клієнта й пристрою, реєстрацію замовлення, призначення майстра, внесення результатів діагностики, додавання робіт і запчастин, зміну статусу, перегляд стану замовлення клієнтом і формування звітів.

Рольовий доступ перевіряється окремо для адміністратора, майстра, керівника та клієнта. Користувач повинен мати доступ лише до функцій, передбачених його роллю [10].

Для кожної перевірки формується тест-кейс. Його структуру наведено в таблиці 4.1.

Таблиця 4.1 – Структура тест-кейсу

Поле	Зміст
Ідентифікатор	Номер тестового сценарію
Назва	Назва перевірки
Передумови	Початковий стан системи
Вхідні дані	Дані для виконання тесту
Дії	Послідовність операцій
Очікуваний результат	Результат, який повинна отримати система
Фактичний результат	Результат виконання тесту
Статус	«Пройдено» або «Не пройдено»

Основні групи тестових сценаріїв наведено в таблиці 4.2.

Таблиця 4.2 – Основні групи тестових сценаріїв

Група	Приклади перевірок
Авторизація	Правильний і неправильний пароль, заборонений доступ
Клієнтські заявки	Коректне подання, порожні поля
Замовлення	Створення, перегляд, редагування, фільтрація
Ремонт	Діагностика, додавання робіт, зміна статусу
Запчастини	Додавання та перевірка кількості
Статус замовлення	Правильний номер, відсутнє замовлення
Звітність	Відображення зведених показників
База даних	Перевірка ключів, унікальності та транзакцій

Критерієм успішного проходження тесту є відповідність фактичного результату очікуваному, правильне збереження даних і дотримання прав доступу.

Послідовність проведення тестування наведено на рисунку 4.1.

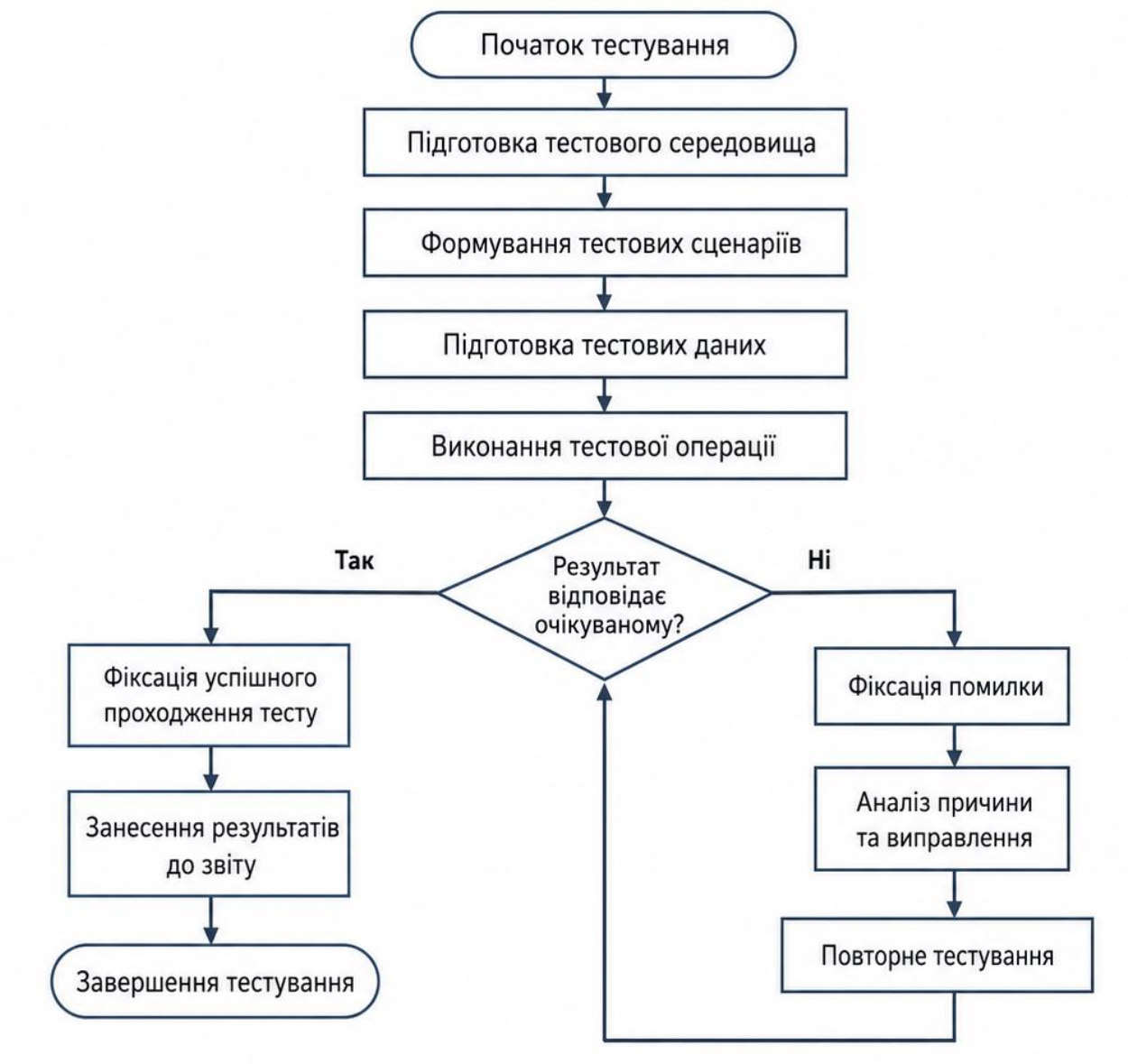

Рисунок 4.1 – Послідовність тестування інформаційної системи

Отже, обрана методика дозволяє перевірити основні функції системи, взаємодію її компонентів, правильність обробки даних і розмежування доступу користувачів.

4.2 Тестування функціональних можливостей системи

Функціональне тестування виконувалося для перевірки основних операцій веб-орієнтованої інформаційної системи обліку замовлень. Під час тестування використовувалися як коректні, так і помилкові вхідні дані, що відповідає підходу перевірки поведінки системи за різними сценаріями використання [23].

Перевірялися такі можливості системи:

– авторизація внутрішніх користувачів;

– подання клієнтської заявки;

– створення клієнта та реєстрація пристрою;

– створення й редагування замовлення;

– призначення майстра;

– внесення результатів діагностики;

– додавання ремонтних робіт і запчастин;

– зміна статусу замовлення;

– перегляд статусу клієнтом;

– формування зведеної інформації для керівника.

Результати основних тестових сценаріїв наведено в таблиці 4.3.

Під час перевірки рольового доступу встановлено, що адміністратор може працювати з клієнтами та замовленнями, майстер — з призначеними ремонтами, а керівник — зі звітною інформацією. Спроба виконання недоступної для ролі операції завершується повідомленням про недостатні права. Така перевірка підтверджує коректність реалізації рольового розмежування доступу [10].

Також було перевірено взаємодію між React, FastAPI та MySQL. Після виконання операцій у клієнтському інтерфейсі сервер коректно обробляв запити, а зміни зберігалися в базі даних. Такий вид перевірки належить до інтеграційного тестування, оскільки оцінює правильність взаємодії окремих компонентів системи [24].

Таблиця 4.3 – Результати функціонального тестування системи

№	Тестова операція	Очікуваний результат	Фактичний результат	Статус
1	Вхід із правильними даними	Відкриття робочої панелі користувача	Робочу панель відкрито	Пройдено
2	Вхід із неправильним паролем	Повідомлення про помилку	Повідомлення відображено	Пройдено
3	Подання коректної заявки	Збереження заявки в базі даних	Заявку збережено	Пройдено
4	Подання заявки з порожніми полями	Заборона надсилання форми	Відображено помилки полів	Пройдено
5	Створення нового замовлення	Формування унікального номера	Замовлення створено	Пройдено
6	Призначення майстра	Збереження майстра в замовленні	Дані оновлено	Пройдено
7	Внесення результатів діагностики	Збереження результатів	Результати збережено	Пройдено
8	Додавання ремонтної роботи	Оновлення переліку робіт і вартості	Дані оновлено	Пройдено
9	Зміна статусу замовлення	Оновлення статусу та історії	Статус та історію оновлено	Пройдено
10	Перегляд статусу за номером	Відображення актуального стану	Статус відображено	Пройдено
11	Пошук неіснуючого замовлення	Повідомлення про відсутність запису	Повідомлення відображено	Пройдено

Отже, основні функціональні можливості системи працюють відповідно до визначених вимог. Критичних помилок, які перешкоджають виконанню базових операцій, під час тестування не виявлено.

4.3 Аналіз результатів тестування

За результатами проведеного тестування встановлено, що розроблена веб-орієнтована інформаційна система виконує основні функції відповідно до визначених вимог. Усі тестові сценарії, наведені в таблиці 4.3, завершилися успішно.

Система коректно обробляє клієнтські заявки, забезпечує створення та супроводження замовлень, призначення майстрів, внесення результатів діагностики, облік ремонтних робіт і запчастин, а також зміну статусів. Дані, введені через клієнтську частину React, правильно передаються до серверної частини FastAPI та зберігаються в базі даних MySQL.

Перевірка рольового доступу підтвердила, що користувачам надаються лише дозволені функції. Адміністратор керує клієнтами й замовленнями, майстер працює з призначеними ремонтами, керівник переглядає зведену інформацію, а клієнт може подати заявку та перевірити статус замовлення.

Під час введення некоректних даних система відображає відповідні повідомлення та не зберігає помилкові записи. Перевірка зв’язків між таблицями також підтвердила правильність збереження пов’язаних даних та історії зміни статусів.

Таким чином, результати тестування підтверджують працездатність розробленої системи та її відповідність поставленим функціональним вимогам. Критичних помилок не виявлено, а система може використовуватися для автоматизації обліку замовлень ремонтної майстерні.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано задачу розробки веб-орієнтованої інформаційної системи обліку замовлень ремонтної майстерні. Створена система призначена для автоматизації основних процесів приймання, супроводження, виконання та завершення замовлень.

У процесі виконання роботи було проаналізовано діяльність ремонтної майстерні та визначено особливості обліку замовлень у цій предметній області. Встановлено, що основними інформаційними об’єктами є клієнти, пристрої, заявки, замовлення, результати діагностики, ремонтні роботи, запчастини, статуси та оплати. Також було розглянуто існуючі програмні рішення й визначено їх переваги та недоліки.

На основі проведеного аналізу сформульовано функціональні вимоги до системи. Передбачено роботу чотирьох основних ролей: клієнта, адміністратора, майстра та керівника. Для кожної ролі визначено окремий набір функцій і рівень доступу до даних.

У роботі розроблено функціональні схеми, алгоритми основних процесів, UML-діаграми взаємодії компонентів і переходів між інтерфейсами. Це дозволило визначити логіку роботи системи, порядок обробки замовлення та взаємодію користувачів із програмним забезпеченням.

Для реалізації системи обрано клієнт-серверну багаторівневу архітектуру. Клієнтську частину реалізовано засобами React, серверну частину — мовою Python із використанням FastAPI, а для збереження даних застосовано MySQL. Обмін даними між клієнтською та серверною частинами організовано через REST API у форматі JSON.

Було спроєктовано структуру реляційної бази даних, яка містить таблиці користувачів, ролей, клієнтів, пристроїв, заявок, замовлень, статусів, історії статусів, результатів діагностики, ремонтних робіт, запчастин та оплат. Зв’язки між таблицями реалізовано за допомогою первинних і зовнішніх ключів.

Користувацький інтерфейс побудовано з урахуванням рольового доступу. Для клієнта передбачено подання заявки та перегляд статусу замовлення. Адміністратор працює з клієнтами, заявками та замовленнями. Майстер вносить результати діагностики й ремонтних робіт, а керівник переглядає зведені показники роботи майстерні.

Проведене тестування підтвердило коректну роботу основних функціональних можливостей системи. Було перевірено авторизацію, створення та редагування замовлень, призначення майстра, внесення результатів діагностики, зміну статусів, облік запчастин, перегляд стану замовлення клієнтом і формування звітної інформації. Критичних помилок під час тестування не виявлено.

Отже, поставлену мету роботи досягнуто, а визначені завдання виконано. Розроблена веб-орієнтована інформаційна система забезпечує централізований облік замовлень, зменшує кількість ручних операцій, спрощує пошук інформації та покращує контроль за виконанням ремонтних робіт.

Подальший розвиток системи може передбачати впровадження автоматичних повідомлень для клієнтів, розширеної аналітики, резервного копіювання даних, інтеграції з платіжними сервісами та розгортання програмного забезпечення на віддаленому сервері.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Documentation. React: The library for web and native user interfaces [Electronic resource]. – Access mode: https://react.dev/ – Date of access: 01.06.2026.
2. Python Documentation [Electronic resource]. – Access mode: https://docs.python.org/ – Date of access: 03.06.2026.
3. MySQL Documentation [Electronic resource]. – Access mode: https://dev.mysql.com/doc/ – Date of access: 05.06.2026.
4. Microsoft Excel [Electronic resource]. – Access mode: https://www.microsoft.com/uk-ua/microsoft-365/excel – Date of access: 06.06.2026.
5. Orderry. Repair Shop Software [Electronic resource]. – Access mode: https://orderry.com/repair-shop-software/ – Date of access: 08.06.2026.
6. RO App. CRM для сервісних центрів: облік замовлень і складу [Electronic resource]. – Access mode: https://roapp.com.ua/ – Date of access: 08.06.2026.
7. Odoo. Inventory management software [Electronic resource]. – Access mode: https://www.odoo.com/app/inventory – Date of access: 07.06.2026.
8. Zoho Inventory. Sales Order Management [Electronic resource]. – Access mode: https://www.zoho.com/inventory/sales-order-management/ – Date of access: 07.06.2026.
9. Sommerville I. Software Engineering. 10th ed. – Boston: Pearson, 2016. – 816 p.
10. Ferraiolo D. F., Kuhn D. R., Chandramouli R. Role-Based Access Control. 2nd ed. – Boston: Artech House, 2007. – 344 p.
11. Nielsen J. Usability Engineering. – San Francisco: Morgan Kaufmann, 1993. – 362 p.
12. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. – Boston: Pearson, 2011. – 960 p.

13. MDN Web Docs. HTTP [Electronic resource]. – Access mode: https://developer.mozilla.org/en-US/docs/Web/HTTP – Date of access: 04.06.2026.

14. Fowler M. Patterns of Enterprise Application Architecture. – Boston: Addison-Wesley, 2002. – 560 p.

15. Date C. J. An Introduction to Database Systems. 8th ed. – Boston: Addison-Wesley, 2004. – 1024 p.

16. FastAPI Documentation [Electronic resource]. – Access mode: https://fastapi.tiangolo.com/ – Date of access: 02.06.2026.

17. Richardson L., Amundsen M., Ruby S. RESTful Web APIs. – Sebastopol: O'Reilly Media, 2013. – 406 p.

18. MDN Web Docs. Web technology for developers [Electronic resource]. – Access mode: https://developer.mozilla.org/en-US/docs/Web – Date of access: 04.06.2026.

19. SQLAlchemy Documentation [Electronic resource]. – Access mode: https://docs.sqlalchemy.org/ – Date of access: 06.06.2026.

20. Visual Studio Code Documentation [Electronic resource]. – Access mode: https://code.visualstudio.com/docs – Date of access: 06.06.2026.

21. Git Documentation [Electronic resource]. – Access mode: https://git-scm.com/docs – Date of access: 06.06.2026.

22. MDN Web Docs. JSON [Electronic resource]. – Access mode: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON – Date of access: 04.06.2026.

23. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. – Hoboken: John Wiley & Sons, 2012. – 256 p.

24. ISO/IEC/IEEE 29119-1:2022. Software and systems engineering — Software testing — Part 1: General concepts. – Geneva: ISO, 2022.

Додаток А – Лістинг модулів програми

Лістинг А.1 – Головний файл серверного застосунку main.py

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

from app.routers import auth_router
from app.routers import clients_router
from app.routers import devices_router
from app.routers import requests_router
from app.routers import orders_router
from app.routers import diagnostics_router
from app.routers import works_router
from app.routers import parts_router
from app.routers import reports_router

app = FastAPI(
    title="Workshop Orders Information System",
    description="Web-oriented information system for repair
workshop order accounting",
    version="1.0.0"
)

origins = [
    "http://localhost:3000",
    "http://localhost:5173"
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["GET", "POST", "PATCH", "DELETE"],
    allow_headers=["*"],
)

@app.get("/")
def root():
    return {
        "message": "Workshop Orders API is running"
    }

app.include_router(auth_router.router, prefix="/api/auth",
tags=["Authorization"])
app.include_router(clients_router.router, prefix="/api/clients",
tags=["Clients"])
app.include_router(devices_router.router, prefix="/api/devices",
tags=["Devices"])
app.include_router(requests_router.router,
prefix="/api/requests", tags=["Requests"])
```

```
app.include_router orders_router.router, prefix="/api/orders",
tags=["Orders"])
app.include_router(diagnostics_router.router,
prefix="/api/diagnostics", tags=["Diagnostics"])
app.include_router(works_router.router, prefix="/api/works",
tags=["Repair works"])
app.include_router(parts_router.router, prefix="/api/parts",
tags=["Parts"])
app.include_router(reports_router.router, prefix="/api/reports",
tags=["Reports"])
```

Лістинг А.2 – Налаштування підключення до бази даних database.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base

DATABASE_URL =
"mysql+pymysql://root:password@localhost:3306/workshop_orders"

engine = create_engine(
    DATABASE_URL,
    pool_pre_ping=True,
    echo=False
)

SessionLocal = sessionmaker(
    autocommit=False,
    autoflush=False,
    bind=engine
)


Base = declarative_base()


def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Функція get_db() використовується як залежність FastAPI. Вона створює сесію підключення до бази даних для кожного запиту, а після завершення обробки запиту закриває її.

Лістинг А.3 – Основні ORM-моделі системи models.py

```
from sqlalchemy import (
    Column,
    Integer,
    String,
    Text,
    DateTime,
    ForeignKey,
    Boolean,
    Numeric
)
from sqlalchemy.orm import relationship
from datetime import datetime

from app.database import Base

class Role(Base):
    __tablename__ = "roles"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(50), unique=True, nullable=False)

    users = relationship("User", back_populates="role")

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    role_id = Column(Integer, ForeignKey("roles.id"),
nullable=False)
    login = Column(String(100), unique=True, nullable=False)
    password_hash = Column(String(255), nullable=False)
    full_name = Column(String(150), nullable=False)
    phone = Column(String(30))
    is_active = Column(Boolean, default=True)
    created_at = Column(DateTime, default=datetime.utcnow)

    role = relationship("Role", back_populates="users")
    assigned_orders = relationship("Order",
back_populates="master")

class Client(Base):
    __tablename__ = "clients"

    id = Column(Integer, primary_key=True, index=True)
    full_name = Column(String(150), nullable=False)
    phone = Column(String(30), nullable=False)
    email = Column(String(100))
    address = Column(String(255))
    notes = Column(Text)
    created_at = Column(DateTime, default=datetime.utcnow)
```

```
    devices = relationship("Device", back_populates="client")
    orders = relationship("Order", back_populates="client")

class Device(Base):
    __tablename__ = "devices"

    id = Column(Integer, primary_key=True, index=True)
    client_id = Column(Integer, ForeignKey("clients.id"),
nullable=False)
    device_type = Column(String(100), nullable=False)
    brand = Column(String(100))
    model = Column(String(100))
    serial_number = Column(String(100))
    equipment = Column(Text)
    condition_description = Column(Text)

    client = relationship("Client", back_populates="devices")
    orders = relationship("Order", back_populates="device")

class Status(Base):
    __tablename__ = "statuses"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), unique=True, nullable=False)
    description = Column(Text)

    orders = relationship("Order", back_populates="status")

class Order(Base):
    __tablename__ = "orders"

    id = Column(Integer, primary_key=True, index=True)
    order_number = Column(String(50), unique=True,
nullable=False)

    client_id = Column(Integer, ForeignKey("clients.id"),
nullable=False)
    device_id = Column(Integer, ForeignKey("devices.id"),
nullable=False)
    master_id = Column(Integer, ForeignKey("users.id"))
    status_id = Column(Integer, ForeignKey("statuses.id"),
nullable=False)

    fault_description = Column(Text, nullable=False)
    accepted_at = Column(DateTime, default=datetime.utcnow)
    planned_completion_at = Column(DateTime)
    completed_at = Column(DateTime)

    labor_cost = Column(Numeric(10, 2), default=0)
    parts_cost = Column(Numeric(10, 2), default=0)
    total_cost = Column(Numeric(10, 2), default=0)
```

```
    notes = Column(Text)
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, default=datetime.utcnow)

    client = relationship("Client", back_populates="orders")
    device = relationship("Device", back_populates="orders")
    master = relationship("User",
back_populates="assigned_orders")
    status = relationship("Status", back_populates="orders")

    diagnostics = relationship("Diagnostic",
back_populates="order")
    works = relationship("RepairWork", back_populates="order")
    status_history = relationship("StatusHistory",
back_populates="order")

class StatusHistory(Base):
    __tablename__ = "status_history"

    id = Column(Integer, primary_key=True, index=True)
    order_id = Column(Integer, ForeignKey("orders.id"),
nullable=False)
    old_status_id = Column(Integer, ForeignKey("statuses.id"))
    new_status_id = Column(Integer, ForeignKey("statuses.id"),
nullable=False)
    changed_by = Column(Integer, ForeignKey("users.id"),
nullable=False)
    comment = Column(Text)
    changed_at = Column(DateTime, default=datetime.utcnow)

    order = relationship("Order",
back_populates="status_history")

class Diagnostic(Base):
    __tablename__ = "diagnostics"

    id = Column(Integer, primary_key=True, index=True)
    order_id = Column(Integer, ForeignKey("orders.id"),
nullable=False)
    master_id = Column(Integer, ForeignKey("users.id"),
nullable=False)
    result = Column(Text, nullable=False)
    recommendation = Column(Text)
    estimated_cost = Column(Numeric(10, 2), default=0)
    created_at = Column(DateTime, default=datetime.utcnow)

    order = relationship("Order", back_populates="diagnostics")

class RepairWork(Base):
    __tablename__ = "repair_works"

    id = Column(Integer, primary_key=True, index=True)
```

```
    order_id = Column(Integer, ForeignKey("orders.id"),
nullable=False)
    master_id = Column(Integer, ForeignKey("users.id"),
nullable=False)
    description = Column(Text, nullable=False)
    cost = Column(Numeric(10, 2), default=0)
    completed_at = Column(DateTime, default=datetime.utcnow)

    order = relationship("Order", back_populates="works")

class Part(Base):
    __tablename__ = "parts"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(150), nullable=False)
    article = Column(String(100))
    quantity = Column(Integer, default=0)
    purchase_price = Column(Numeric(10, 2), default=0)
    sale_price = Column(Numeric(10, 2), default=0)

class OrderPart(Base):
    __tablename__ = "order_parts"

    id = Column(Integer, primary_key=True, index=True)
    order_id = Column(Integer, ForeignKey("orders.id"),
nullable=False)
    part_id = Column(Integer, ForeignKey("parts.id"),
nullable=False)
    quantity = Column(Integer, nullable=False)
    unit_price = Column(Numeric(10, 2), nullable=False)


class Payment(Base):
    __tablename__ = "payments"

    id = Column(Integer, primary_key=True, index=True)
    order_id = Column(Integer, ForeignKey("orders.id"),
nullable=False)
    amount = Column(Numeric(10, 2), nullable=False)
    payment_method = Column(String(50))
    payment_status = Column(String(50), default="Очікує оплату")
    paid_at = Column(DateTime)
```

Лістинг А.4 – Схеми даних schemas.py

```
from pydantic import BaseModel, Field
from typing import Optional, List
from datetime import datetime
from decimal import Decimal

class TokenResponse(BaseModel):
```

```
    access_token: str
    token_type: str
    role: str
    full_name: str

class LoginRequest(BaseModel):
    login: str = Field(..., min_length=3)
    password: str = Field(..., min_length=4)

class ClientCreate(BaseModel):
    full_name: str
    phone: str
    email: Optional[str] = None
    address: Optional[str] = None
    notes: Optional[str] = None

class ClientResponse(BaseModel):
    id: int
    full_name: str
    phone: str
    email: Optional[str]

    class Config:
        from_attributes = True

class DeviceCreate(BaseModel):
    client_id: int
    device_type: str
    brand: Optional[str] = None
    model: Optional[str] = None
    serial_number: Optional[str] = None
    equipment: Optional[str] = None
    condition_description: Optional[str] = None

class DeviceResponse(BaseModel):
    id: int
    client_id: int
    device_type: str
    brand: Optional[str]
    model: Optional[str]

    class Config:
        from_attributes = True

class RequestCreate(BaseModel):
    client_name: str
    phone: str
    device_type: str
    fault_description: str

class OrderCreate(BaseModel):
    client_id: int
    device_id: int
```

```
    master_id: Optional[int] = None
    status_id: int
    fault_description: str
    planned_completion_at: Optional[datetime] = None
    notes: Optional[str] = None

class OrderUpdate(BaseModel):
    master_id: Optional[int] = None
    status_id: Optional[int] = None
    fault_description: Optional[str] = None
    planned_completion_at: Optional[datetime] = None
    labor_cost: Optional[Decimal] = None
    parts_cost: Optional[Decimal] = None
    total_cost: Optional[Decimal] = None
    notes: Optional[str] = None

class OrderResponse(BaseModel):
    id: int
    order_number: str
    client_id: int
    device_id: int
    master_id: Optional[int]
    status_id: int
    fault_description: str
    total_cost: Decimal
    created_at: datetime

    class Config:
        from_attributes = True

class DiagnosticCreate(BaseModel):
    order_id: int
    result: str
    recommendation: Optional[str] = None
    estimated_cost: Optional[Decimal] = 0

class RepairWorkCreate(BaseModel):
    order_id: int
    description: str
    cost: Decimal

class PartCreate(BaseModel):
    name: str
    article: Optional[str] = None
    quantity: int
    purchase_price: Decimal
    sale_price: Decimal

class OrderPartCreate(BaseModel):
    order_id: int
    part_id: int
    quantity: int
    unit_price: Decimal
```

```
class StatusChangeRequest(BaseModel):
    new_status_id: int
    comment: Optional[str] = None

class ReportSummary(BaseModel):
    active_orders: int
    completed_orders: int
    canceled_orders: int
    total_income: Decimal
```

Схеми дозволяють відокремити структуру даних від логіки їх обробки. Наприклад, схема OrderCreate використовується під час створення нового замовлення, а OrderResponse — для повернення даних про замовлення клієнтській частині.

Лістинг А.5 – Модуль авторизації auth.py

```
from datetime import datetime, timedelta
from typing import Optional

from jose import jwt, JWTError
from passlib.context import CryptContext
from fastapi import HTTPException, status

from app.models import User

SECRET_KEY = "workshop_secret_key"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 60

pwd_context = CryptContext(schemes=["bcrypt"],
deprecated="auto")

def verify_password(plain_password: str, password_hash: str) ->
bool:
    return pwd_context.verify(plain_password, password_hash)

def get_password_hash(password: str) -> str:
    return pwd_context.hash(password)

def create_access_token(data: dict, expires_delta:
Optional[timedelta] = None):
    to_encode = data.copy()

    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
```

```
        expire = datetime.utcnow() +
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)

    to_encode.update({"exp": expire})

    encoded_jwt = jwt.encode(
        to_encode,
        SECRET_KEY,
        algorithm=ALGORITHM
    )

    return encoded_jwt

def authenticate_user(db, login: str, password: str):
    user = db.query(User).filter(User.login == login).first()

    if not user:
        return None

    if not verify_password(password, user.password_hash):
        return None

    if not user.is_active:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Користувач заблокований"
        )

    return user

def decode_access_token(token: str):
    try:
        payload = jwt.decode(
            token,
            SECRET_KEY,
            algorithms=[ALGORITHM]
        )

        user_id = payload.get("sub")
        role = payload.get("role")

        if user_id is None:
            raise HTTPException(
                status_code=status.HTTP_401_UNAUTHORIZED,
                detail="Неправильний токен доступу"
            )

        return {
            "user_id": int(user_id),
            "role": role
        }

    except JWTError:
```

```
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Не вдалося перевірити токен доступу"
        )
```

Лістинг А.6 – Залежності для перевірки користувача dependencies.py

```
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from sqlalchemy.orm import Session

from app.database import get_db
from app.auth import decode_access_token
from app.models import User


oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/api/auth/login")


def get_current_user(
    token: str = Depends(oauth2_scheme),
    db: Session = Depends(get_db)
):
    token_data = decode_access_token(token)

    user = db.query(User).filter(User.id ==
token_data["user_id"]).first()

    if user is None:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Користувача не знайдено"
        )

    return user


def require_roles(*roles):
    def role_checker(current_user: User =
Depends(get_current_user)):
        if current_user.role.name not in roles:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="Недостатньо прав для виконання операції"
            )
        return current_user

    return role_checker
```

Функція require_roles() використовується в маршрутах REST API для перевірки доступу. Якщо роль користувача не відповідає дозволеним ролям, сервер повертає помилку з кодом 403 Forbidden.

Лістинг А.7 – Маршрут авторизації auth_router.py

```
from datetime import timedelta

from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session

from app.database import get_db
from app.schemas import LoginRequest, TokenResponse
from app.auth import authenticate_user, create_access_token

router = APIRouter()

@router.post("/login", response_model=TokenResponse)
def login(
    request: LoginRequest,
    db: Session = Depends(get_db)
):
    user = authenticate_user(
        db=db,
        login=request.login,
        password=request.password
    )

    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Неправильний логін або пароль"
        )

    access_token = create_access_token(
        data={
            "sub": str(user.id),
            "role": user.role.name
        },
        expires_delta=timedelta(minutes=60)
    )

    return {
        "access_token": access_token,
        "token_type": "bearer",
        "role": user.role.name,
        "full_name": user.full_name
    }
```

Лістинг А.8 – Маршрути для роботи з клієнтами clients_router.py

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session

from app.database import get_db
from app.models import Client
from app.schemas import ClientCreate, ClientResponse
from app.dependencies import require_roles


router = APIRouter()


@router.get("/", response_model=list[ClientResponse])
def get_clients(
    search: str | None = None,
    db: Session = Depends(get_db),
    current_user=Depends(require_roles("administrator",
"manager", "master"))
):
    query = db.query(Client)

    if search:
        query = query.filter(
            Client.full_name.like(f"%{search}%")
        )

    return query.order_by(Client.created_at.desc()).all()


@router.post("/", response_model=ClientResponse)
def create_client(
    client_data: ClientCreate,
    db: Session = Depends(get_db),
    current_user=Depends(require_roles("administrator"))
):
    client = Client(
        full_name=client_data.full_name,
        phone=client_data.phone,
        email=client_data.email,
        address=client_data.address,
        notes=client_data.notes
    )

    db.add(client)
    db.commit()
    db.refresh(client)

    return client
```

```
@router.get("/{client_id}", response_model=ClientResponse)
def get_client_by_id(
    client_id: int,
    db: Session = Depends(get_db),
    current_user=Depends(require_roles("administrator",
"manager", "master"))
):
    client = db.query(Client).filter(Client.id ==
client_id).first()

    if not client:
        raise HTTPException(
            status_code=404,
            detail="Клієнта не знайдено"
        )

    return client
```

Маршрут GET /api/clients дозволяє отримати список клієнтів, а маршрут POST /api/clients — створити нового клієнта. Створення клієнтів доступне лише адміністратору.

Лістинг А.9 – Маршрути для роботи з пристроями devices_router.py

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session

from app.database import get_db
from app.models import Device, Client
from app.schemas import DeviceCreate, DeviceResponse
from app.dependencies import require_roles

router = APIRouter()

@router.post("/", response_model=DeviceResponse)
def create_device(
    device_data: DeviceCreate,
    db: Session = Depends(get_db),
    current_user=Depends(require_roles("administrator"))
):
    client = db.query(Client).filter(Client.id ==
device_data.client_id).first()

    if not client:
        raise HTTPException(
            status_code=404,
            detail="Клієнта не знайдено"
        )

    device = Device(
```

```
        client_id=device_data.client_id,
        device_type=device_data.device_type,
        brand=device_data.brand,
        model=device_data.model,
        serial_number=device_data.serial_number,
        equipment=device_data.equipment,
        condition_description=device_data.condition_description
    )

    db.add(device)
    db.commit()
    db.refresh(device)

    return device

@router.get("/client/{client_id}",
response_model=list[DeviceResponse])
def get_client_devices(
    client_id: int,
    db: Session = Depends(get_db),
    current_user=Depends(require_roles("administrator",
"master", "manager"))
):
    return db.query(Device).filter(Device.client_id ==
client_id).all()
```

Лістинг А.10 – Маршрут подання клієнтської заявки requests_router.py

```
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from datetime import datetime

from app.database import get_db
from app.schemas import RequestCreate
from sqlalchemy import Column, Integer, String, Text, DateTime
from app.database import Base


class ClientRequest(Base):
    __tablename__ = "requests"

    id = Column(Integer, primary_key=True, index=True)
    client_name = Column(String(150), nullable=False)
    phone = Column(String(30), nullable=False)
    device_type = Column(String(100), nullable=False)
    fault_description = Column(Text, nullable=False)
    request_status = Column(String(50), default="Нова")
    created_at = Column(DateTime, default=datetime.utcnow)


router = APIRouter()
```

```
@router.post("/")
def create_request(
    request_data: RequestCreate,
    db: Session = Depends(get_db)
):
    request = ClientRequest(
        client_name=request_data.client_name,
        phone=request_data.phone,
        device_type=request_data.device_type,
        fault_description=request_data.fault_description,
        request_status="Нова"
    )

    db.add(request)
    db.commit()
    db.refresh(request)

    return {
        "message": "Заявку успішно створено",
        "request_id": request.id
    }

@router.get("/")
def get_requests(
    db: Session = Depends(get_db)
):
    return
db.query(ClientRequest).order_by(ClientRequest.created_at.desc()
).all()
```

Лістинг А.11 – Сервіс створення замовлення order_service.py

```
from datetime import datetime

from app.models import Order, StatusHistory

def generate_order_number(db):
    current_year = datetime.utcnow().year

    last_order = (
        db.query(Order)
        .order_by(Order.id.desc())
        .first()
    )

    if last_order is None:
        next_number = 1
    else:
        next_number = last_order.id + 1
```

```
    return f"ORD-{current_year}-{next_number:05d}"

def create_order_service(db, order_data, current_user):
    order_number = generate_order_number(db)

    order = Order(
        order_number=order_number,
        client_id=order_data.client_id,
        device_id=order_data.device_id,
        master_id=order_data.master_id,
        status_id=order_data.status_id,
        fault_description=order_data.fault_description,
        planned_completion_at=order_data.planned_completion_at,
        notes=order_data.notes
    )

    db.add(order)
    db.commit()
    db.refresh(order)

    history = StatusHistory(
        order_id=order.id,
        old_status_id=None,
        new_status_id=order.status_id,
        changed_by=current_user.id,
        comment="Створення нового замовлення"
    )

    db.add(history)
    db.commit()

    return order
```