

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
КОМП'ЮТЕРНОГО СЕРВІСУ
SOFTWARE ENGINEERING
FOR COMPUTER SERVICE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-112

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ГОГОЛЕНКО Д. С.

(ПРИЗВИЩЕ та ініціали)

Керівник ГОФМАН Є. О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ДЬЯЧУК Т. С.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ГОГОЛЕНКА Дмитра Сергійович
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Інженерія програмного забезпечення комп'ютерного сервісу.
Software Engineering for Computer Service
керівник проєкту (роботи) к.т.н., доцент, ГОФМАН Євгеній Олександрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГОФМАН Є. О., доцент		
Нормоконтроль	КАЛІНІНА М. В., асистент		

7. Дата видачі завдання «20» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Дмитро ГОГОЛЕНКО
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Євгеній ГОФМАН
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра: 137 с., 9 табл., 28 рис., 3 дод., 20 джерел.

АВТЕНТИФІКАЦІЯ, БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ЗАМОВЛЕННЯ, КОМП'ЮТЕРНИЙ СЕРВІС, ОБЛІК КЛІЄНТІВ, POSTGRESQL, REACT, REST API.

Об'єкт дослідження – процес інженерії програмного забезпечення обліку замовлень, клієнтів, працівників, послуг і запасних частин у комп'ютерному сервісі.

Предмет дослідження – методи та програмні засоби для розроблення інформаційних систем на базі архітектури клієнт-сервер.

Мета роботи – підвищення рівня автоматизації та впорядкованості обліку роботи комп'ютерного сервісу шляхом розроблення вебзастосунку для ведення клієнтів, працівників, замовлень, послуг, запасних частин і пов'язаних операцій.

Матеріали, методи та технічні засоби. Використано методи структурного та об'єктно-орієнтованого проектування, реляційне моделювання даних, клієнт-серверну архітектуру, REST API та методи функціонального тестування. Програмну реалізацію виконано з використанням JavaScript, React, Vite, Bootstrap, Node.js, Express, PostgreSQL, TypeORM, Zod, JWT і refresh-механізму. Розроблення проводилося у середовищі Visual Studio Code з використанням npm і Git.

Результати. Розроблено вебзастосунок «Сервісний центр», який забезпечує автентифікацію користувачів, розмежування прав адміністратора і користувача, ведення довідників клієнтів, працівників, послуг і запасних частин, оформлення та редагування замовлень, додавання послуг і запасних частин до заявки, пошук і сортування записів, підказки у формах, друк

квитанції, перегляд зведених подань і статистики. Побудовано модель бази даних PostgreSQL, серверний API на Express та клієнтський інтерфейс на React.

Висновки. Мету роботи досягнуто: створено програмний продукт, що централізує дані сервісної майстерні, зменшує залежність від паперових журналів і розрізнених таблиць, підвищує прозорість стану замовлень та забезпечує базовий рівень захисту даних. Проведені перевірки підтвердили працездатність основних сценаріїв, достатню швидкодію для умов малої майстерні та коректність базових механізмів безпеки.

Галузь використання – внутрішній облік роботи комп'ютерного сервісу, ремонтної майстерні або навчального стенду з розроблення вебзастосунків.

Економічна ефективність. Використання застосунку скорочує час на пошук інформації про клієнтів, замовлення й виконані роботи, знижує ризик помилок при веденні заявок і підвищує контрольованість роботи персоналу.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor:
137 pages, 9 tables, 28 figures, 3 appendixes, 20 sources.

AUTHENTICATION, COMPUTER SERVICE, DATABASE, ORDER,
POSTGRES SQL, REACT, REPAIR ACCOUNTING, REST API, WEB
APPLICATION.

Object of research – the software engineering process for managing orders, customers, employees, services, and spare parts in a computer repair service.

Subject of research – methods and software tools for developing information systems based on the client-server architecture.

The purpose of the work – to increase the level of automation and organization of accounting processes in a computer service center by developing a web application for managing clients, employees, orders, services, spare parts, and related operations.

Materials, methods, and technical means: structural and object-oriented design methods, relational data modeling, client-server architecture, REST API, and functional testing methods were used. The software was implemented using JavaScript, React, Vite, Bootstrap, Node.js, Express, PostgreSQL, TypeORM, Zod, JWT, and a refresh-session mechanism. Development was carried out in Visual Studio Code using npm and Git.

Results. The “Service Center” web application was developed. It provides user authentication, separation of administrator and user permissions, management of clients, employees, services and spare parts, creation and editing of orders, adding services and spare parts to an order, searching and sorting records, form suggestions, receipt printing, viewing read-only database views, and statistics. A PostgreSQL database model, an Express server API, and a React client interface were implemented.

Conclusions. The work's objective was achieved: a software product was created that centralizes service center data, reduces dependence on paper journals and separate spreadsheets, improves the transparency of order statuses, and provides a basic level of data protection. The performed checks confirmed the operability of the main scenarios, sufficient performance for a small workshop, and correct operation of basic security mechanisms.

Field of application – internal accounting processes of a computer service center, repair workshop, or educational stand for web application development.

Economic efficiency. The application reduces the time required to search for information about clients, orders, and completed work, decreases the risk of errors when processing service requests, and improves staff activity control.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	11
Вступ	12
1 Аналіз предметної області	14
1.1 Короткий опис досліджуваної предметної області	14
1.2 Огляд існуючих рішень проблеми	15
1.2.1 Застосунок RemOnline.....	15
1.2.2 Застосунок HubSpot CRM.....	16
1.2.3 Застосунок Google Таблиці	18
1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів.....	19
1.3 Опис основних вимог до програми.....	19
1.3.1 Визначення функціональних вимог.....	19
1.3.2 Нефункціональні вимоги.....	20
1.4 Постановка завдання роботи.....	21
1.5 Висновки за розділом 1	22
2 Розробка архітектури програми.....	23
2.1 Структура системи	23
2.1.1 Основні компоненти системи.....	23
2.1.2 Взаємодія компонентів системи	23
2.2 Функціонування системи	24
2.2.1 Основні процеси системи	24
2.2.2 Логіка роботи системи.....	25
2.3 Вибір мови програмування та інструментарію для розробки програми.....	25
2.3.1 Мова програмування.....	25
2.3.2 Середовище розробки.....	27
2.3.3 База даних.....	28

2.4 Вимоги до технічних і програмних засобів.....	29
2.4.1 Необхідні технічні засобів для розробки та експлуатації.....	29
2.4.2 Вимоги до апаратного та програмного забезпечення.....	29
2.5 Висновки за розділом 2	29
3 Розробка програми.....	30
3.1 Архітектура системи.....	30
3.1.1 Структурна схема системи	31
3.2 Функціонування системи	33
3.2.1 Функціональна схема	33
3.2.2 Опис процесу обміну даними у системі	34
3.2.3 Сценарії використання	35
3.3 Опис модулів системи.....	37
3.3.1 Модуль автентифікації та керування користувачами.....	37
3.3.2 Модуль замовлень	40
3.3.3 Модуль позицій замовлення.....	42
3.3.4 Модуль довідників послуг і запасних частин.....	44
3.3.5 Модуль обліку клієнтів і персоналу	46
3.3.6 Модуль подань і статистики.....	47
3.3.7 Модуль інфраструктури вебзастосунку.....	48
3.3.8 Проєктування бази даних	49
3.4 Висновки за розділом 3	51
4 Експлуатація, тестування та експериментальне дослідження програми.....	53
4.1 Опис застосування програми.....	53
4.1.1 Призначення та область застосування.....	53
4.1.2 Методи реалізації та класи вирішуваних завдань.....	54
4.1.3 Обмеження застосування.....	54
4.1.4 Мінімальна конфігурація технічних засобів	55
4.2 Інструкція по роботі з програмою.....	56
4.2.1 Встановлення програми.....	56
4.2.2 Запуск програми.....	58

	10
4.2.3 Методика роботи з програмою	58
4.3 Приклад роботи користувача з програмою	59
4.4 Тестування роботи програми	60
4.4.1 Функціональне тестування	60
4.4.2 Тестування продуктивності	63
4.4.3 Тестування безпеки	64
4.4.4 Аналіз користувацького досвіду	64
4.5 Рекомендації щодо використання	65
4.6 Плани з розвитку	66
Висновки	67
Перелік джерел посилання	69
Додаток А Технічне завдання	71
Додаток Б Текст програми	80
Додаток В Слайди презентації	130

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CRUD	– Create, Read, Update, Delete;
CSS	– Cascading Style Sheets;
HTML	– HyperText Markup Language;
HTTP	– HyperText Transfer Protocol;
HTTPS	– HyperText Transfer Protocol Secure;
JSON	– JavaScript Object Notation;
JWT	– JSON Web Token;
ORM	– Object-Relational Mapping;
REST	– Representational State Transfer;
SPA	– Single Page Application;
SQL	– Structured Query Language;
TLS	– Transport Layer Security;
UI	– User Interface;
URL	– Uniform Resource Locator;
ЗІП	– запасні інструменти та приладдя;
СУБД	– система управління базами даних.

ВСТУП

Сучасний стан розвитку інформаційних технологій характеризується активним переходом організацій від паперових журналів і розрізнених електронних таблиць до централізованих вебсистем обліку. Для комп'ютерних сервісів така автоматизація є особливо важливою, оскільки щоденна робота майстерні пов'язана з прийомом техніки, веденням клієнтів, призначенням відповідальних майстрів, фіксацією статусів ремонту, обліком послуг і використаних запасних частин. За відсутності єдиного сховища даних зростає ризик втрати інформації, дублювання записів, помилок у статусах замовлень і складності швидкого пошуку історії ремонту.

Проблема автоматизації обліку роботи сервісної майстерні є актуальною для малих і середніх підприємств, які потребують доступного програмного засобу без надмірної залежності від сторонніх хмарних платформ. Існуючі комерційні рішення надають широкий набір функцій, однак часто мають абонентську вартість, обмежену гнучкість зміни бізнес-логіки або зберігають дані у зовнішній інфраструктурі. Універсальні CRM-системи не повністю відповідають ремонтному циклу, а електронні таблиці не забезпечують достатнього рівня цілісності даних, рольового доступу та контрольованої роботи з пов'язаними сутностями.

Проблема дослідження полягає в необхідності створення вебзастосунку, який забезпечує централізоване ведення даних комп'ютерного сервісу, підтримує типові сценарії роботи з клієнтами, замовленнями, послугами й запасними частинами, надає зручний інтерфейс для пошуку, сортування, заповнення форм і друку квитанції, а також реалізує базові механізми автентифікації та розмежування прав. Огляд існуючих підходів показує, що для навчального проєкту доцільно розробити власне рішення на базі відкритих технологій React, Node.js, Express і PostgreSQL.

Об'єктом дослідження є процес автоматизації обліку роботи комп'ютерного сервісу.

Предметом дослідження є вебзастосунок для ведення клієнтів, працівників, замовлень, послуг, запасних частин і пов'язаних операцій з використанням клієнт-серверної архітектури та реляційної бази даних.

Мета роботи – підвищення рівня автоматизації та впорядкованості обліку роботи комп'ютерного сервісу шляхом розроблення вебзастосунку «Сервісний центр».

Для досягнення поставленої мети в роботі вирішувалися такі завдання:

- проаналізувати предметну область комп'ютерного сервісу, існуючі рішення та вимоги до програмного продукту;
- розробити архітектуру вебзастосунку, модель бази даних, структуру серверного API та клієнтського інтерфейсу;
- програмно реалізувати модулі автентифікації, адміністрування користувачів, ведення довідників, замовлень, позицій замовлення, подань і статистики;
- перевірити працездатність системи, її швидкодію, базові механізми безпеки та зручність використання.

Методика дослідження включає аналіз предметної області та аналогів, структурне проєктування архітектури, реляційне моделювання даних, програмну реалізацію клієнтської та серверної частин, а також функціональне, продуктивнісне й безпекове тестування на навчальному стенді. Для реалізації використано JavaScript, React, Vite, Bootstrap, Node.js, Express, PostgreSQL, TypeORM, Zod, JWT, npm і Git.

Розроблений вебзастосунок може бути використаний для внутрішнього обліку в невеликому комп'ютерному сервісі або як навчальний прототип інформаційної системи сервісної майстерні. Практична цінність результатів полягає у централізації даних, скороченні часу на пошук інформації, зменшенні кількості помилок під час оформлення заявок і підвищенні прозорості стану замовлень для персоналу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Короткий опис досліджуваної предметної області

Діяльність комп'ютерного сервісу охоплює прийом техніки від клієнтів, діагностику, оформлення замовлень на ремонт або обслуговування, закупівлю й списання запасних частин, розподіл навантаження між майстрами та інформування замовника про стан робіт. У малих і середніх майстернях оперативні дані нерідко фіксують у паперових журналах, розрізнених таблицях або переписують між змінами – це підвищує ризик втрати записів, дублювання номенклатури, суперечностей у статусах замовлень і ускладнює відповіді на типові питання: що саме робили з пристроєм, які комплектуючі вже змонтовані, хто виконував роботу.

Основні поняття предметної області, які мають бути відображені в обліковій системі:

- клієнт – контрагент, що передав техніку або замовив послугу;
- замовлення (заявка) – документований цикл робіт із конкретною одиницею техніки, датами, статусом виконання, відповідальним майстром;
- послуга – номенклатурна одиниця наданих робіт;
- запасна частина (ЗП) – складська позиція, що може бути зарезервована або витрачена в межах замовлення;
- зв'язок замовлення з послугами та ЗП – перелік фактичних робіт і матеріалів у складі одного замовлення;
- представлення (подання) даних – зведені подання для швидкого перегляду даних без ручного збирання з кількох джерел.

Сучасні тенденції включають перехід від «паперу» до вебдоступу з єдиної бази, попит на розмежування доступу за ролями (керівництво, приймальня, майстер), інтеграція з власною інфраструктурою замість повної залежності від стороннього SaaS. Для України актуальним лишається також можливість локального або гібридного розгортання, контроль над даними клієнтів і гнучка адаптація під внутрішні правила майстерні.

1.2 Огляд існуючих рішень проблеми

Нижче розглянуто три варіанти обліку в сервісі, з якими порівнюється розроблюваний застосунок – хмарна платформа RemOnline [1], хмарна HubSpot CRM [2] і ведення записів у Google Таблицях [3].

1.2.1 Застосунок RemOnline

RemOnline – це спеціалізоване комерційне хмарне рішення, що функціонує за моделлю SaaS і орієнтоване на автоматизацію бізнес-процесів у серійних та ремонтних підприємствах. Дана система розроблена для комплексного ведення операційного обліку замовлень, моніторингу життєвого циклу ремонтних статусів у реальному часі, координації дій робочого персоналу, а також для оптимізації складського обліку деталей та витратних матеріалів.

Переваги:

- швидкий старт без власного сервера;
- готові екрани під ремонтний цикл;
- підтримка з боку постачальника;
- мобільні клієнти.

Недоліки:

- обмежена свобода зміни бізнес-логіки й структури даних під індивідуальний сценарій;
- дані клієнтів розміщені зовнішньо хмарою, що для частини підприємств є чутливим питанням;
- відкритий програмний код відсутній.

Вигляд інтерфейсу застосунку RemOnline продемонстровано на рисунку 1.1.

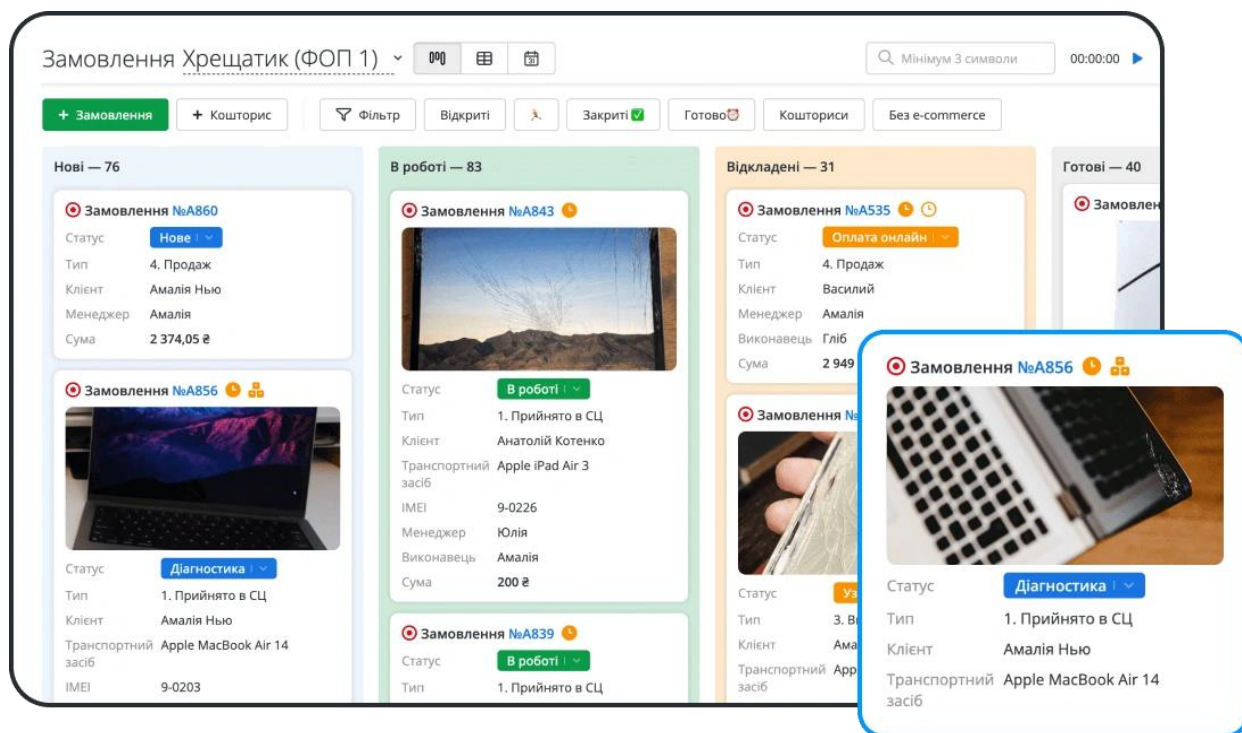


Рисунок 1.1 –Інтерфейс RemOnline [1]

1.2.2 Застосунок HubSpot CRM

HubSpot CRM – хмарна платформа керування взаємодією з клієнтами від компанії HubSpot Inc. (США). Система охоплює облік контактів і компаній, воронки продажів із картками угод, управління завданнями, електронну пошту та вбудований модуль звітності. Базовий рівень доступний безкоштовно, проте з обмеженим набором функцій і допустимим обсягом даних; розширені можливості та сторонні інтеграції відкриваються лише за платною підпискою. За своєю концепцією платформа орієнтована на маркетинг і процеси продажів (B2B/B2C) – операційний цикл ремонтної майстерні виходить за межі її цільового призначення.

Переваги:

- зручні картки угод і етапи воронки;
- історія комунікацій з клієнтом;
- мобільний застосунок;
- публічний API для інтеграцій;

– для малого обсягу даних – можливість старту без плати за ліцензію CRM.

Недоліки:

- немає готової схеми «ремонт ПК» – замовлення з послугами, ЗІП і статусом доведеться вигадувати через угоди та додаткові поля, тож простий сценарій «прийняв – відремонтував – видав» стає громіздким;
- облік складу й залишків запчастин у системі не передбачений як у спеціалізованого сервісу;
- усі дані зберігаються в хмарі постачальника (зазвичай за кордоном);
- для майстерні, якій потрібен лише внутрішній облік, інтерфейс і платні опції часто надлишкові – багато функцій про продажі й маркетинг, мало про цех.

Вигляд інтерфейсу застосунку HubSpot CRM продемонстровано на рисунку 1.2.

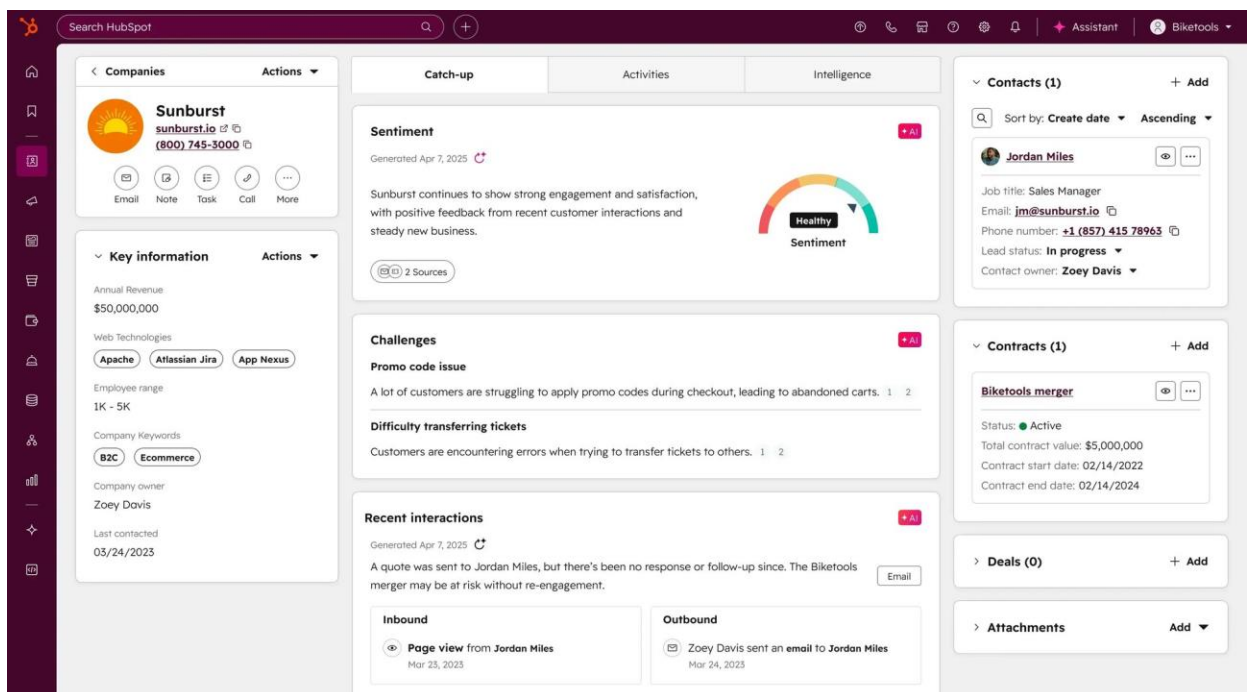


Рисунок 1.2 – Інтерфейс HubSpot CRM [2]

1.2.3 Застосунок Google Таблиці

Google Таблиці – це потужний, безкоштовний хмарний вебзастосунок для створення, редагування та аналізу електронних таблиць. Сервіс дає змогу швидко опрацьовувати дані безпосередньо у веббраузері без встановлення спеціального програмного забезпечення.

Переваги:

- низький поріг входу, достатньо облікового запису Google;
- спільне редагування й історія змін у хмарі;
- доступ з браузера та з телефона;
- формули, фільтри й зведені таблиці.

Недоліки:

- відсутність правил цілісності правилами цілісності;
- відсутність моделі зв'язків «клієнт – замовлення – послуги – ЗП»;
- проблема одночасного редагування;
- мала продуктивність;
- публічний API потребує окремої розробки.

Вигляд інтерфейсу застосунку Google Таблиці продемонстровано на рисунку 1.3.

Tag	Name	Company	Work Email	Title	Contact Type	Work Phone	Work Website	Address	City	Zip
Followup	Thomas Liao	ozerflex	thomas@ozerflex	Marketing Manager	Potential Customer	562-391-2345	www.ozerflex.co	224 Main St	San Francisco	90
Hot	Lillian Soto	Fixfase	lillian@fixfase.co	Sales Rep	Potential Customer	562-391-2346	www.fixfase.co	543 Senwood St	New York	90
Soon	Violette Gatewood	Namdrill	violette@namdrill	Sales Operator	Current Customer	562-391-2347	www.namdrill.co	2430 Grand Corr	Sucker Brook	28
Soon	Mary Garcia	Saltace	mary@saltace.co	Community Manager	Other	562-391-2348	www.saltace.co	1275 Harvest Be	Hills and Dales	33
Hot	Cameron Lyle	Moveplane	cameron@move	Designer	Current Customer	562-391-2349	www.moveplane	6329 Velvet Cloc	Nitta Yuma	74
Followup	Mildred Noriega	Finware	mildred@finware	CEO	Potential Customer	562-391-2350	www.finware.co	1903 Quiet Willo	Muenster	27
Upsale	Melody Estes	Driplectronics	melody@driplec	Executive Assistant	Other	562-391-2351	www.driplectron	3446 Jagged Wz	Platinum	34

Рисунок 1.3 – Інтерфейс Google Таблиці [3]

1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів

Для визначення переваг та недоліків розроблюваної системи проведемо порівняння із застосунками RemOnline, HubSpot SRM та Google Таблиці за ключовими критеріями. Результати аналізу наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння підходів до обліку в сервісі комп'ютерної техніки

Критерій	RemOnline	HubSpot CRM	Google Таблиці	Сервісний центр (розроблюваний продукт)
Орієнтація на предметну область	Висока	Низька	Низька	Висока
Зв'язки між даними	Так	Обмежено	Частково	Так
Ролі доступу	Так	Так	Частково	Так
Витрати	Абонплата	Абонплата	Без витрат	Сервер
API для інтеграцій	Обмежено	Так	Відсутній	Так
Зведення даних	Обмежено	Так	Обмежено	Так

1.3 Опис основних вимог до програми

1.3.1 Визначення функціональних вимог

Для вебзастосунку «Сервісний центр» доцільно виділити такі групи функціональних вимог:

а) облік сутностей сервісу:

1) ведення довідників: клієнти, співробітники, замовлення, послуги, запасні частини, рядки «замовлені послуги» та «замовлені запасні частини»;

2) зведена інформація та статистика;

3) операції перегляду, створення, оновлення й видалення із валідацією вхідних даних;

б) користувачі та безпека:

1) вхід за електронною поштою та паролем, зберігання пароля у вигляді криптографічного гешу;

2) рольовий доступ: адміністратор має права реєстрації та видалення облікових записів користувачів;

3) перевірка ролей та ключів безпеки перед виконанням операцій;

в) клієнтський інтерфейс:

1) робота у браузері;

2) табличний перегляд записів із пошуком і сортуванням, підказки при заповненні форм, а також друк квитанції з картки замовлення.

1.3.2 Нефункціональні вимоги

Система повинна відповідати таким нефункціональним вимогам:

а) вимоги до безпеки:

1) автентифікація користувачів;

2) безпечне зберігання паролів;

3) розмежування прав доступу;

4) захищена передача даних;

б) вимоги до надійності:

1) перевірка коректності даних;

2) обробка помилок без збоїв системи;

3) збереження цілісності даних;

в) вимоги до продуктивності:

1) швидке відкриття списків і форм;

2) збереження записів без помітних затримок;

- 3) підтримка кількох одночасних користувачів;
- г) вимоги до користувацького інтерфейсу:
 - 1) робота через веббраузер;
 - 2) пошук і сортування записів;
 - 3) підказки у формах;
 - 4) зрозумілі повідомлення користувачу;
- д) вимоги до експлуатації:
 - 1) робота у локальній мережі;
 - 2) адміністрування облікових записів;
 - 3) резервне копіювання даних;
- е) вимоги до масштабованості:
 - 1) можливість додавання нових модулів;
 - 2) підтримка зростання обсягу даних;
 - 3) придатність до подальшого розвитку;
- ж) вимоги до сумісності:
 - 1) підтримка сучасних браузерів;
 - 2) узгоджений обмін даними;
 - 3) придатність для навчального та робочого середовища.

1.4 Постановка завдання роботи

Розроблення вебзастосунку «Сервісний центр» спрямоване на подолання розриву між універсальними комерційними сервісами та потребою майстерні у спеціалізованій системі.

Основні завдання роботи:

- проєктування архітектури клієнт-серверної системи;
- реалізація клієнтської та серверної частини системи;
- забезпечення безпечної авторизації та збереження даних;
- тестування розробленої системи;
- оформлення документації.

Цілі роботи:

- зменшення часу на пошук інформації;
- зниження помилок обліку;
- підвищення прозорості статусів ремонту;
- забезпечення базової інформаційної безпеки для даних майстерні.

1.5 Висновки за розділом 1

Предметна область обліку роботи комп'ютерного сервісу залишається актуальною через поширеність пристроїв, потребу в швидкому ремонті й обслуговуванні та неефективність розрізнених ручних методів ведення записів.

Огляд типових альтернатив показав, що комерційні хмарні продукти дають готовий функціонал, але накладають вартість і залежність від постачальника, універсальні CRM вимагають значної адаптації, електронні таблиці не забезпечують достатньої централізації й безпеки для зростаючого обсягу замовлень.

Розроблюваний вебзастосунок «Сервісний центр» орієнтований на усунення цих недоліків за рахунок предметної моделі, централізованого сховища даних, розмежування ролей, клієнт-серверної архітектури та можливості подальшого розширення інтерфейсу. Сформульовані вимоги узгоджуються з технічним завданням і створюють основу для подальших розділів пояснювальної записки (архітектура, реалізація, тестування).

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Структура системи

2.1.1 Основні компоненти системи

Архітектура вебзастосунку «Сервісний центр» базується на клієнт-серверній моделі з розподілом на три логічні рівні (рівень представлення, рівень прикладної логіки та рівень даних) та REST-подібному протоколі взаємодії [4].

Рівень представлення (клієнт):

- сторінки – відображення інтерфейсу користувача;
- компоненти – спільні багаторазові елементи інтерфейсу.

Рівень прикладної логіки (сервер застосунку):

- шлюз обробки запитів – приймає вхідні http-запити, забезпечує перевірку та валідацію вхідних даних;
- модуль безпеки – перевіряє маркери доступу в заголовках запитів та обмежує доступ до ресурсів на основі ролей користувачів.

Рівень даних:

- база даних – забезпечує збереження таблиць бізнес-даних, облікових записів користувачів та активних сесій оновлення безпеки.

2.1.2 Взаємодія компонентів системи

Взаємодія рівня представлення з рівнем прикладної логіки:

- сторінки реагують на дії користувача та ініціюють відправку асинхронних запитів у стандартизованому форматі;
- шлюз обробки запитів та проміжне ПЗ сервера перевіряють структуру даних і наявність прав доступу;
- результат виконання операції повертається клієнту для оновлення стану інтерфейсу.

Взаємодія шару прикладної логіки з шаром даних:

- сервіси управління сутностями взаємодіють із базою даних через об'єктно-реляційне відображення або службові запити;
- безпека даних посилюється за рахунок динамічного встановлення ролі сеансу СУБД перед виконанням запитів відповідно до прав поточного користувача.

Для входу передбачається видача короткоживучого access token і встановлення refresh token у httpOnly-кукі. Оновлення сесії виконується окремим запитом, після чого сервер інвалідує стару refresh-сесію й видає нову пару маркерів. Вихід супроводжується відкликанням сесії в базі та очищенням кукі.

2.2 Функціонування системи

2.2.1 Основні процеси системи

Функціонування системи можна описати через основні процеси, які відбуваються під час використання програми.

Процес автентифікації:

- введення пошти й пароля;
- перевірка облікових даних;
- видача маркера доступу;
- підтвердження подальших запитів.

Процес адміністрування користувачів:

- створення облікових записів;
- перегляд списку користувачів;
- видалення облікових записів;
- заборона доступу без ролі адміністратора.

Процес ведення замовлень і довідників:

- перегляд записів;
- створення та редагування даних;

- додавання послуг і ЗП;
- зміна статусу замовлення.

Процес обробки помилок:

- відхилення некоректних даних;
- повідомлення про конфлікти;
- приховування внутрішніх деталей.

2.2.2 Логіка роботи системи

Логіка розроблюваної системи спирається на такі принципи:

- єдине джерело правди: дані зберігаються в БД, клієнт отримує актуальний стан через API;
- захищені маршрути на сервері: перевірка токенів і ролей;
- гешування паролів;
- керування сесіями на основі токенів;
- принцип ролей та прав;
- контракт API: узгоджена структура тіла запиту і відповіді;
- розширюваність: можливість розширення функціоналу всіх рівнів системи без повної переробки;
- реактивне оновлення: інтерфейс автоматично оновлюється при зміні даних.

2.3 Вибір мови програмування та інструментарію для розробки програми

2.3.1 Мова програмування

Для розроблюваного вебзастосунку «Сервісний центр» планується застосувати мову JavaScript [5]: на клієнті – разом із JSX у фреймворку React [6] і маршрутизацією React Router [7], на сервері – у середовищі Node.js [8].

Обґрунтування вибору зводиться до таких причин:

- єдиний мовний стек клієнта й сервера полегшує узгодженість структур даних, назв полів і підходів до серіалізації в REST API;
- вбудована асинхронна модель природно підходить для мережових запитів і неблокувального I/O в Node.js;
- екосистема npm [9] надає зрілі бібліотеки для Express [10], валідації Zod [11], JWT [12], ORM TypeORM [13] і роботи з PostgreSQL [14], скорочує час реалізації застосунку;
- поєднання React і JavaScript відповідає вимогам до сучасного браузерного клієнта.

Обмеженням JavaScript є слабка статична типізація, що збільшує ризик помилок, які статичний аналіз виявив би раніше. Динамічна природа мови вимагає дисципліни в тестуванні.

Порівняння варіантів мовних стеків для подібного вебпроєкту наведено в табл. 2.1.

Таблиця 2.1 – Порівняння підходів до реалізації вебзастосунку

Критерій	JavaScript (Node.js + React)	Python (Django + JS)	C# (ASP.NET Core + SPA)
Єдність мови клієнта й сервера	Висока	Низька	Низька або часткова
Екосистема пакетів	Дуже висока	Висока	Висока
Асинхронне I/O	Висока	Висока	Висока
Статична типізація	Низька	Низька	Висока
Поріг входу для розробки	Низький	Середній	Середній
Швидкість розробки інтерфейсу (UI)	Дуже висока	Середня	Середня
Швидкість розробки MVP	Висока	Середня	Низька

2.3.2 Середовище розробки

Як основне інтегроване середовище для реалізації проєкту планується Visual Studio Code [15]. Runtime серверної частини – Node.js LTS. Збірка клієнтських модулів – Vite [16]. Керування залежностями – npm (або сумісний менеджер). Контроль версій – Git.

Обґрунтування вибору редактора:

- розширення для JavaScript/JSX, ESLint, інтеграції з Git і вбудований термінал;
- легка адаптація під стек Vite + React + Node.js;
- можливість налагодження застосунків на Node.js через конфігурацію або розширення;
- відсутність плати за ліцензію та широке поширення серед розробників;
- низька ресурсоемність порівняно з важкими IDE повного циклу для інших платформ.

Недоліками обраного середовища розробки є необхідність встановлення додаткових розширень та налаштування для підтримки налагодження та профілювання.

Порівняння середовищ розробки наведено в табл. 2.2.

Таблиця 2.2 – Порівняння середовищ розробки

Критерій	Visual Studio Code	JetBrains WebStorm	Eclipse IDE
Підтримка JS / React	Висока	Висока	Середня
Налагодження Node.js	Так	Так	Частково
Вартість ліцензування	Безкоштовно	Обмежено	Безкоштовно
Ресурсоемність	Низька	Висока	Висока
Інтеграція з Git	Вбудована	Вбудована	Вбудована

2.3.3 База даних

Як СУБД для розроблюваного застосунку обрано PostgreSQL – це безкоштовна об'єктно-реляційна система керування базами даних (СКБД) з відкритим початковим кодом.

Обґрунтування вибору:

- повна реляційна модель і цілісність даних;
- подання та складні SQL-запити;
- гнучкі ролі СУБД;
- стабільні драйвери та інтеграція з TypeORM;
- придатність для локального та серверного розгортання.

Недоліком є потреба в окремій інстанції сервера СУБД і базових навичках адміністрування (резервне копіювання, оновлення, моніторинг).

Порівняння СУБД наведено в табл. 2.3.

Таблиця 2.3 – Порівняння СУБД

Критерій	PostgreSQL	MySQL / MariaDB	SQLite
Транзакції та цілісність	Висока	Висока	Висока
Подання (views)	Сильна підтримка	Сильна підтримка	Підтримується
Ролі та права доступу	Гнучко	Гнучко	Обмежено
Серверний процес СУБД	Так	Так	Ні
Підтримка тригерів	Дуже висока	Середня	Низька
Складні аналітичні запити	Відмінна	Добра	Обмежена
Паралельне виконання запитів	Підтримується	Обмежено	Не підтримується

2.4 Вимоги до технічних і програмних засобів

2.4.1 Необхідні технічні засоби для розробки та експлуатації

Для розробки потрібна робоча станція з об'ємом RAM, достатнім для одночасної роботи редактора, локального сервера Vite, Node.js і клієнта СУБД. Для експлуатації достатньо одного серверного вузла для Node.js та PostgreSQL і робочих місць персоналу з браузером.

2.4.2 Вимоги до апаратного та програмного забезпечення

Апаратні вимоги узгоджуються з п. А.3.4 технічного завдання:

- ПК користувача з дисплеєм не нижче 1280×720;
- сервер – x86-64, від 4–8 Гб RAM;
- диск не менше 20 Гб.

Програмні вимоги:

- ОС Windows / Linux / macOS на робочих місцях;
- браузер останніх стабільних поколінь на клієнті;
- установлені Node.js LTS, PostgreSQL на сервері;
- доступ до портів HTTP/HTTPS для API.

2.5 Висновки за розділом 2

У розділі обґрунтовано розроблювану архітектуру вебзастосунку «Сервісний центр»: виділено клієнт на React, сервер на Node.js/Express із валідацією Zod і JWT + refresh, PostgreSQL + TypeORM як шар даних із поданнями для читання. Описано основні процеси і принципи роботи застосунку. У підрозділі 2.3 подано вибір JavaScript, середовища розробки та PostgreSQL з обґрунтуванням, описом переваг і обмежень і порівняльними таблицями 2.1-2.3.

3 РОЗРОБКА ПРОГРАМИ

3.1 Архітектура системи

Архітектура вебзастосунку «Сервісний центр» базується на трирівневій клієнт–серверній моделі: рівень представлення (SPA у браузері на React), рівень прикладної логіки та REST API (Node.js, Express) і рівень даних (PostgreSQL). Обмін по HTTP у форматі JSON, автентифікація за JWT і механізм оновлення сесії (refresh-токен у httpOnly-кукі) забезпечують чіткий розподіл відповідальності між компонентами, централізовану перевірку прав на сервері й зручне подальше розширення функціональності. Схема взаємодії компонентів системи зображена на рисунку 3.1.

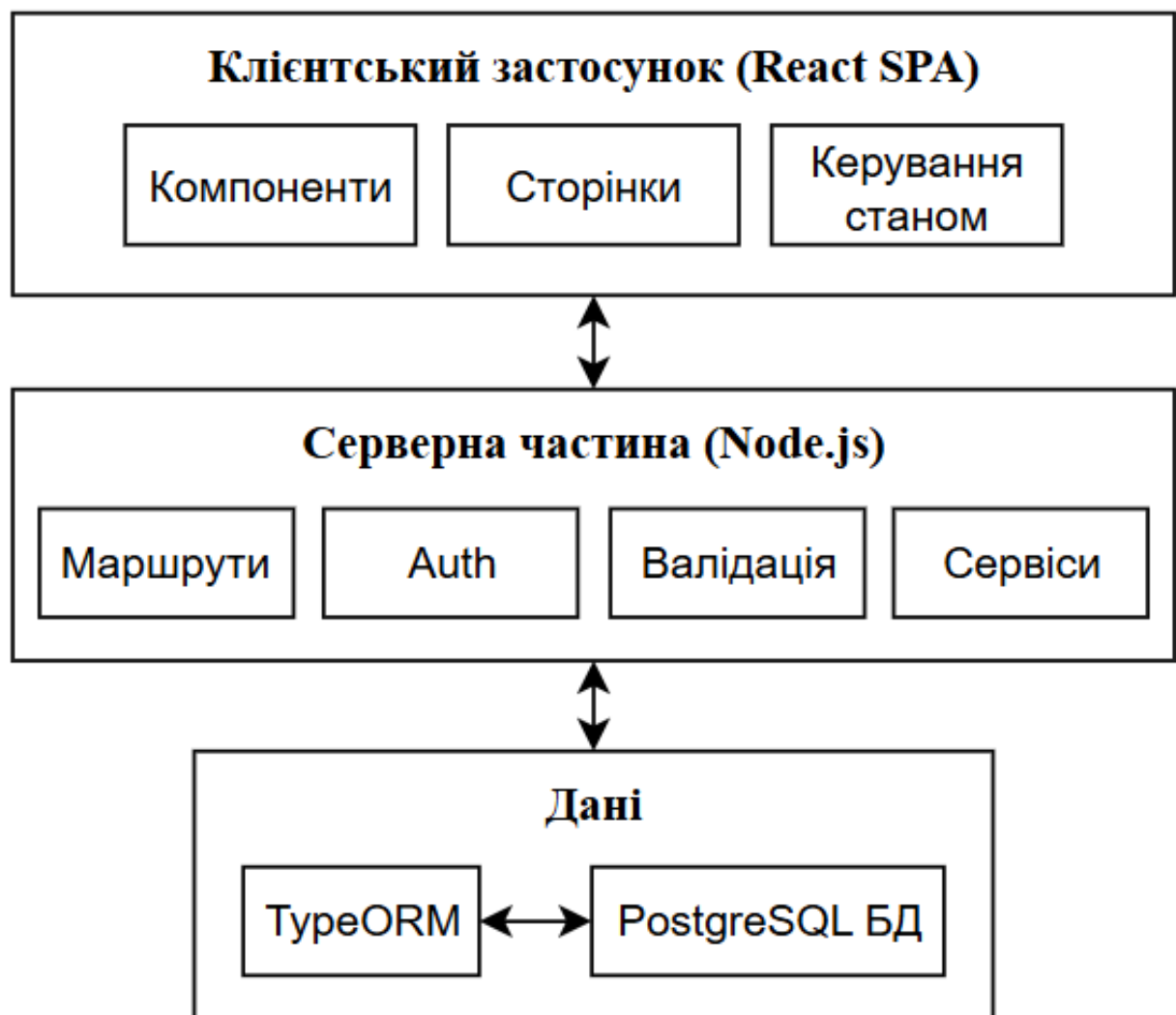


Рисунок 3.1 – Схема взаємодії компонентів системи

Рівень представлення:

- сторінки;
- спільні компоненти;
- оформлення на базі Bootstrap 5 і react-bootstrap [17];
- контекст автентифікації та захищені маршрути;
- централізоване формування запитів з axios [18];
- збірка та середовище розробки: Vite, React Router.

Рівень прикладної логіки:

- middleware перевірки JWT і ролей, валідація даних Zod;
- узагальнені сервіси BaseService та ReadOnlyBaseService;
- маршрути входу, оновлення й завершення сесії;
- реєстрація користувачів адміністратором;
- періодичне очищення прострочених refresh-сесій (node-cron).

Рівень даних:

- таблиці предметної області;
- подання (views) для зведення даних та аналітики;
- облікові записи користувачів та сесії;
- опис сутностей для TypeORM.

3.1.1 Структурна схема системи

Проект організовано за принципом функціональної декомпозиції, де код розділено на логічні модулі відповідно до їх призначення.

Структурну схему системи зображено на рисунку 3.2.

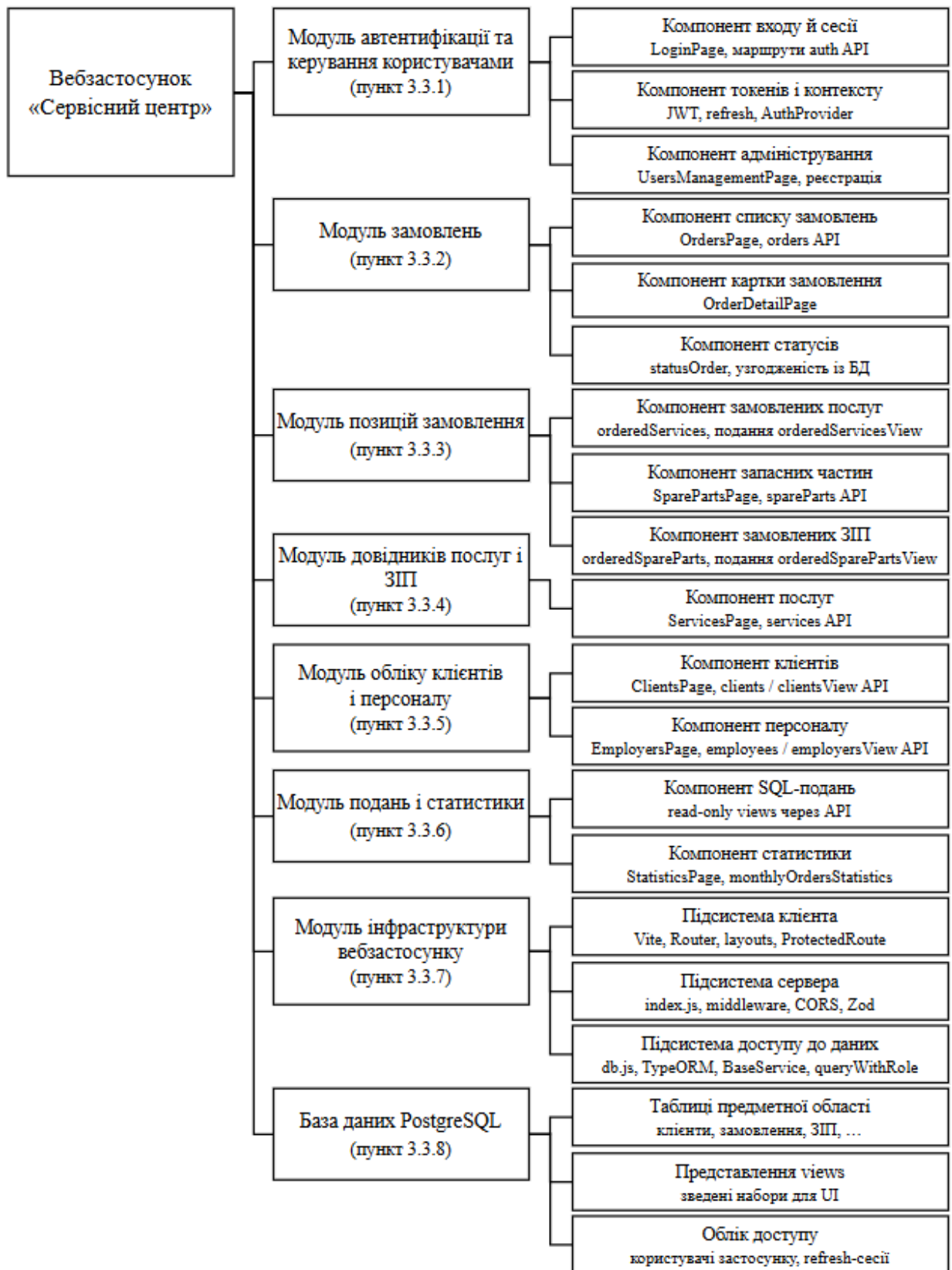


Рисунок 3.2 – Структурна схема системи

3.2 Функціонування системи

3.2.1 Функціональна схема

Вебзастосунок «Сервісний центр» функціонує як клієнт–серверна система: користувач працює у браузері з односторінковим інтерфейсом (React), а облік даних і бізнес-правила зосереджені на сервері (Express, PostgreSQL). Логічні модулі, відображені на структурній схемі (рис. 3.2), реалізуються узгодженими екранами клієнта та маршрутами REST API; усі зміни стану сутностей відображаються в інтерфейсі після відповіді сервера в форматі JSON.

Функціональна схема (рис. 3.3) показує основні зв'язки між користувачем, інтерфейсом, функціональними модулями, серверним API та реляційною базою даних.

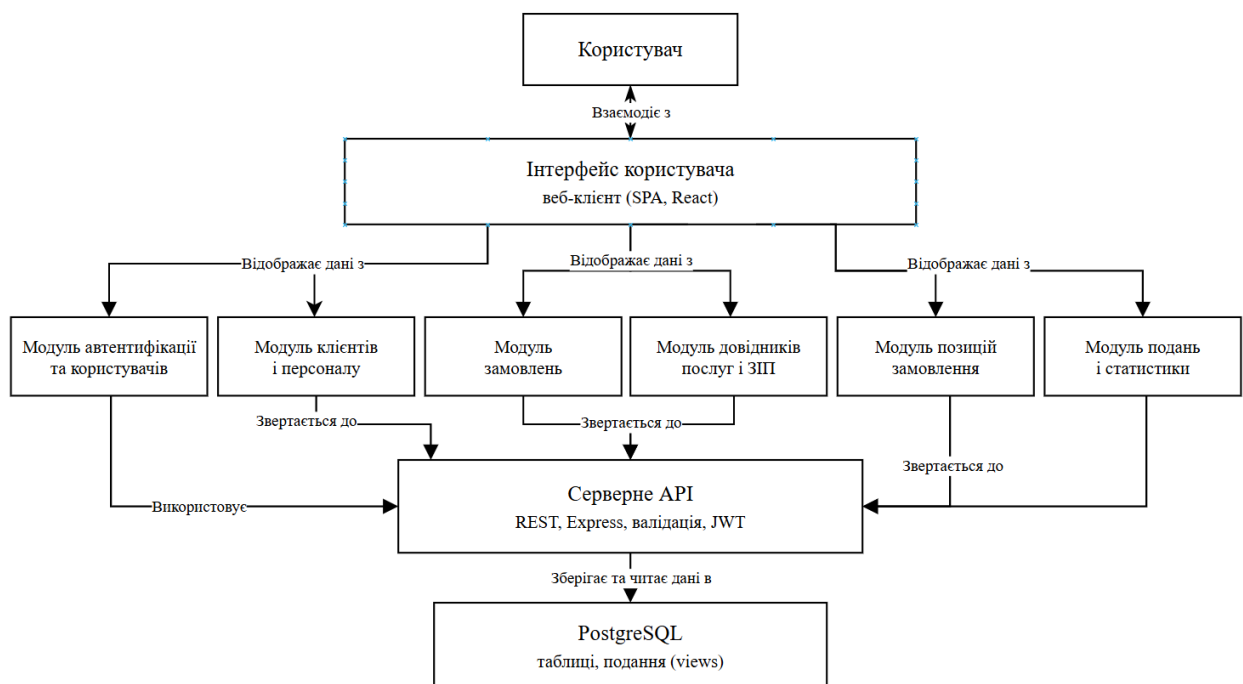


Рисунок 3.3 – Функціональна схема розробленої програми

Через інтерфейс користувач оперує функціями логічних модулів (див. структурну схему, рис. 3.2). Запити до даних і перевірка прав виконуються на сервері через HTTP API; сервер читає й оновлює дані у базі даних.

3.2.2 Опис процесу обміну даними у системі

Процес обміну даними організовано за принципом запит–відповідь по протоколу HTTP: клієнт робить запит до сервера та після підтвердження змін сервером оновлює відображення через механізм стану React.

Схема процесу обміну даними у системі зображена на рисунку 3.4.

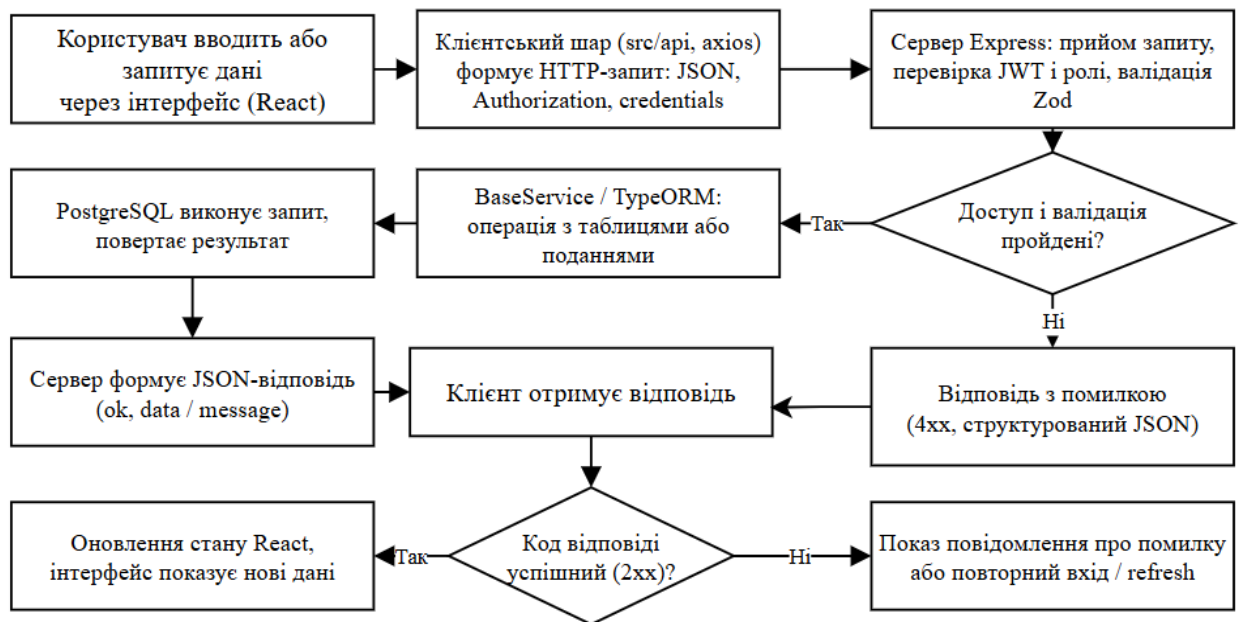


Рисунок 3.4 – Схема процесу обміну даними у системі

Основні етапи потоку даних:

- введення або вибір дій користувачем у вебінтерфейсі (форми, таблиці, перехід між сторінками);
- формування HTTP-запиту на клієнті: модулі каталогу `src/api` (`axios`) додають маркер доступу в заголовок `Authorization`, за потреби передають куки для оновлення сесії, серіалізують тіло в `JSON`;
- прийом запиту на сервері `Express`: журналювання, перевірка `JWT` і ролі для захищених маршрутів, розбір `JSON`-тіла;
- валідація вхідних даних за схемами `Zod`; у разі помилки – відповідь з кодом `VALIDATION_ERROR` і деталями для відображення в інтерфейсі;

Сценарій "Вхід до системи":

- користувач відкриває сторінку автентифікації вебзастосунку;
- вводяться електронна пошта та пароль облікового запису;
- сервер перевіряє надані дані, формує токен та refresh-сесію;
- після успішної автентифікації користувач отримує доступ до робочого інтерфейсу відповідно до призначеної ролі.

Сценарій "Керування обліковими записами":

- адміністратор переходить до розділу керування користувачами;
- створюється новий обліковий запис або видаляється існуючий;
- для запису задається роль, що визначає доступні функції системи;
- після збереження змін список облікових записів оновлюється в інтерфейсі.

Сценарій "Ведення довідників клієнтів, персоналу, запчастин та послуг":

- користувач відкриває відповідний довідник клієнтів, персоналу, запчастин або послуг і за потреби застосовує пошук або сортування;
- у картці запису вводяться або уточнюються дані;
- система виконує перевірку заповнених полів і передає запит на сервер;
- після успішного збереження таблиця довідника відображає актуальний стан даних.

Сценарій "Робота із замовленням":

- користувач переглядає список замовлень, знаходить потрібний запис через пошук або сортує таблицю за колонками;
- у картці замовлення уточнюються реквізити заявки, відповідальний майстер і поточний статус виконання; під час заповнення використовуються підказки;
- зміни передаються через API та зберігаються у базі даних;
- інтерфейс оновлює відомості про замовлення без ручного повторного введення даних, після чого за потреби друкується квитанція.

Сценарій "Додавання послуг і запасних частин до замовлення":

- користувач відкриває перелік позицій у межах конкретного замовлення;
- до заявки додається виконана послуга або використана запасна частина;
- сервер перевіряє коректність зв'язків між замовленням, номенклатурою та кількісними показниками;
- після підтвердження операції картка замовлення відображає оновлений склад робіт і матеріалів.

Сценарій "Перегляд статистики та зведених даних":

- користувач переходить до розділу статистики або зведених подань;
- система виконує запит до агрегованих даних, сформованих на рівні бази;
- у відповідь користувач отримує таблиці й підсумкові показники щодо замовлень, клієнтів, майстрів та виконаних операцій;
- отримані відомості застосовуються для поточного контролю роботи сервісного центру.

3.3 Опис модулів системи

3.3.1 Модуль автентифікації та керування користувачами

3.3.1.1 Алгоритм входу та підтримки сесії користувача

Після заповнення форми дані передаються на сервер; у разі успіху клієнт отримує маркер доступу, а довгоживуча сесія фіксується у захищеній кукі та в таблиці refresh-сесій. Подальші запити містять заголовок Authorization; за відмови через прострочений маркер виконується звернення до ендпоінта оновлення сесії та повтор запиту. Завершення роботи ініціює відкликання refresh-сесії та очищення локального стану.

Узагальнена послідовність кроків зображена на рисунку 3.6.

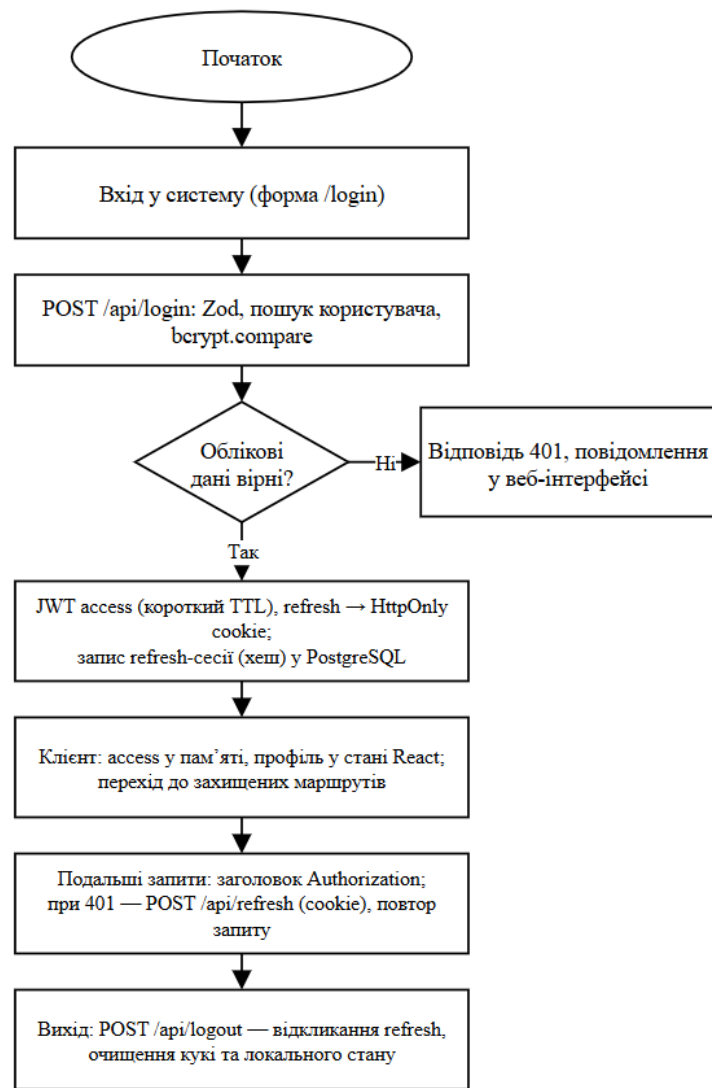


Рисунок 3.6 – Алгоритм входу та підтримки сесії користувача

3.3.1.2 Функціональність модуля

Модуль забезпечує наступні функції:

- вхід у систему за електронною поштою та паролем (перевірка пароля через bcrypt [19]);
- видача пари JWT: access token у тілі відповіді, refresh token у HttpOnly-кукі з обмеженням шляху та атрибутів sameSite/secure у продакшені;
- оновлення access token за дійсною refresh-сесією, ротація запису сесії в базі;
- вихід із системи (відкликання сесії, очищення куки);

- отримання профілю поточного користувача;
- реєстрація нового користувача (лише роль адміністратора), перегляд списку користувачів, видалення облікового запису з обмеженням.

3.3.1.3 Реалізація та використані технології

Серверна частина модуля побудована на Node.js та Express. Для автентифікації й керування користувачами передбачено маршрути `/api/login`, `/api/refresh`, `/api/logout`, `/api/me`, `/api/register`, `/api/users`, `/api/users/:id`. Обробка запитів спирається на `cookie-parser`, `jsonwebtoken` і `bcrypt`, а перевірка тіл запитів та параметрів виконується за допомогою `Zod`.

Refresh-сесії зберігаються в PostgreSQL у вигляді хешів токенів, тому службові дані не фіксуються у відкритому вигляді. На клієнті використано React-контекст автентифікації, модуль `client/src/api/auth.js` та перехоплювачі `axios` у `client/src/api/client.js`, які додають Bearer-токен до захищених запитів і повторюють звернення після оновлення сесії. Інтерфейс представлено сторінкою входу (рис. 3.7) та адміністративною сторінкою керування користувачами (рис. 3.8).

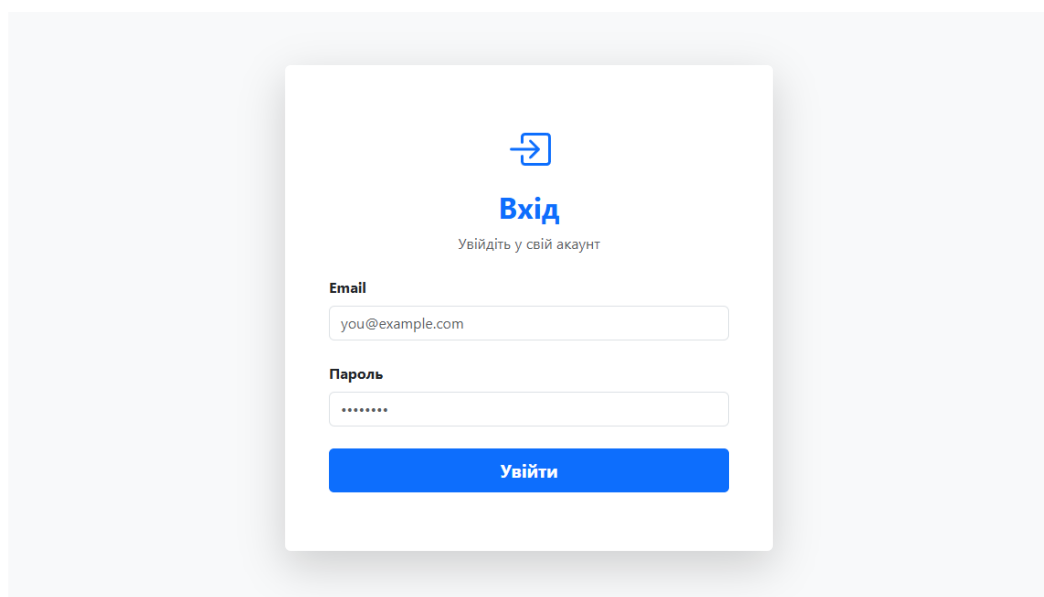


Рисунок 3.7 – Сторінка входу в систему

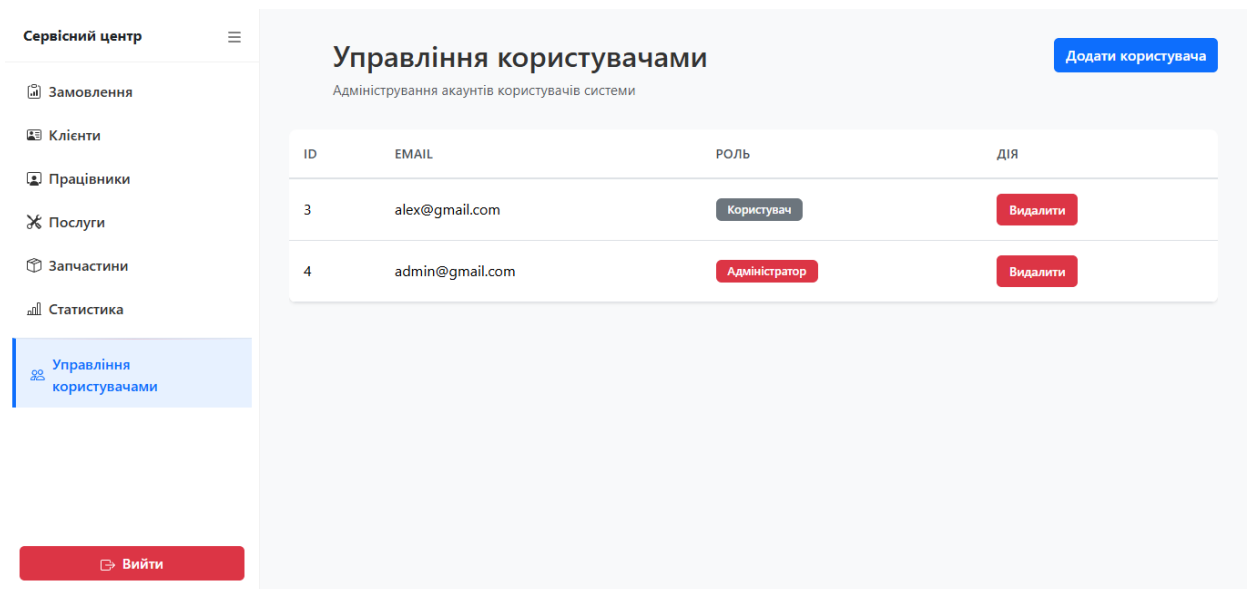


Рисунок 3.8 – Сторінка керування користувачами (роль адміністратора)

3.3.2 Модуль замовлень

3.3.2.1 Алгоритм додавання нового замовлення

Користувач відкриває сторінку списку замовлень і переходить до створення нової заявки.

У формі заповнюються основні реквізити: клієнт, назва або тип пристрою, опис несправності, відповідальний майстер, дата створення, орієнтовна вартість і початковий статус.

Після надсилання форми клієнтська частина формує POST-запит до `/api/orders`, а сервер перевіряє JWT, права доступу та коректність даних за схемою `ordersCrudZod`.

Якщо перевірка проходить успішно, новий запис зберігається в таблиці `orders`, а список заявок в інтерфейсі оновлюється. У разі помилки користувач отримує повідомлення про некоректно заповнені поля або іншу причину відхилення запиту.

Блок-схема алгоритму наведена на рисунку 3.9.

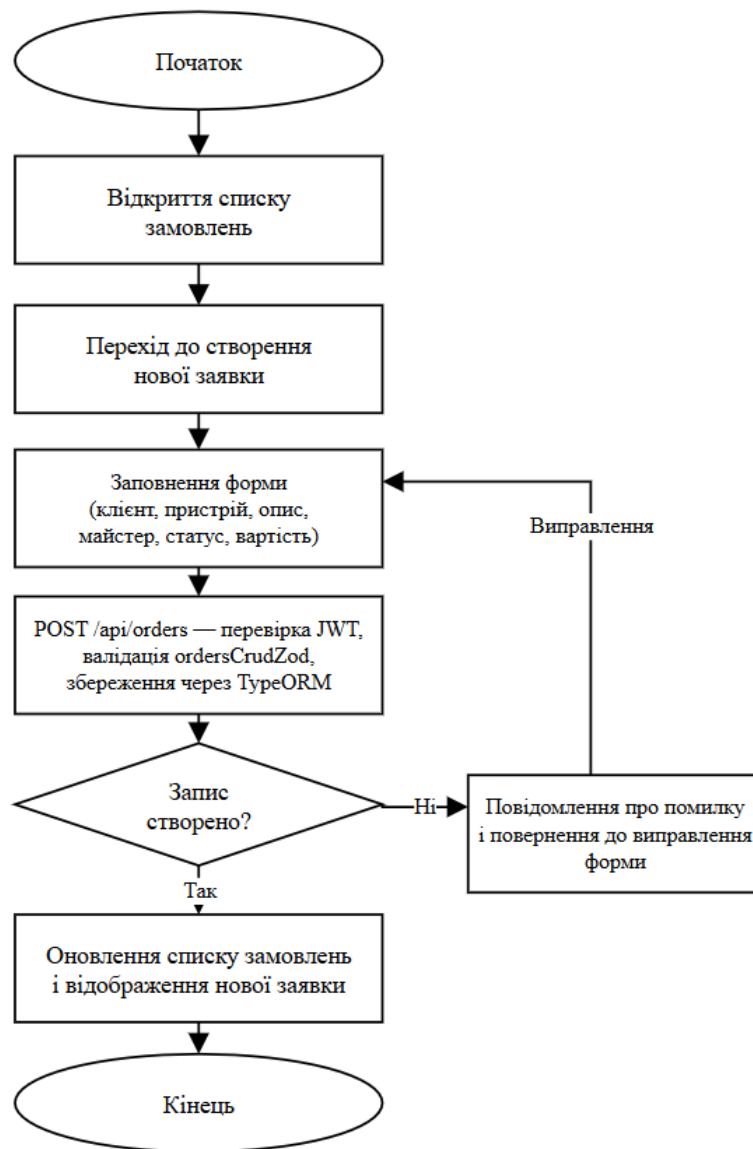


Рисунок 3.9 – Алгоритм додавання нового замовлення

3.3.2.2 Функціональність модуля

Модуль забезпечує наступні функції:

- перегляд списку замовлень (у тому числі через подання orders-view для зручних колонок) із пошуком і сортуванням;
- перегляд і редагування картки замовлення: клієнт, відповідальний майстер, статус, дати, опис техніки та інші атрибути згідно з моделлю;
- підказки при виборі пов'язаних записів у формах;
- друк квитанції зі сторінки деталей замовлення;
- навігація зі списку до картки за маршрутом /orders/:idOrder.

3.3.2.3 Реалізація та використані технології

Робота із замовленнями виконується через маршрути `/api/orders`. Вхідні дані перевіряються за CRUD-схемою `ordersCrudZod`, а доступ до ресурсу захищено JWT і ролями `admin` або `app_user`.

У клієнтській частині модуль реалізовано сторінкою `OrdersPage` (рис. 3.10). Запити до сервера зібрано в API-модулі `orders.js`, а перехід від списку до картки замовлення організовано засобами `react-router-dom`.

ID ЗАМ.	ID КЛ.	КЛІЄНТ	ТЕЛЕФОН	ПРИСТРІЙ	ОПИС	ID М.	МАЙСТЕР	СТВОРЕНО	ОРІЄНТ. ЗАВЕРШЕННЯ	ОРІЄН ВАРТІС
12	2	Олена Кравчук	+380971000002	Samsung NP350	Потрібна діагностика та попередня оцінка ремонту.	2	Марина Петренко	10.05.2026, 12:00:00	10.05.2026, 13:00:00	300 ₴
11	1	Андрій Шевченко	+380971000001	ПК Intel Core i5	Видалення шкідливого ПЗ та налаштування драйверів.	1	Олексій Коваленко	06.05.2026, 16:10:00	—	800 ₴
10	10	Ілона Климчук	+380971000010	HP ProBook 450 G7	Ноутбук не завантажує операційну систему.	4	Ірина Савчук	02.05.2026, 10:35:00	02.05.2026, 13:35:00	3 850 ₴
9	9	Юрій Гончар	+380971000009	ASUS TUF Gaming F15	Очікується SSD для заміни несправного накопичувача.	5	Артем Мельник	27.04.2026, 15:00:00	—	3 300 ₴
8	8	Катерина Лисенко	+380971000008	Lenovo ThinkCentre M720	Заміна оперативної пам'яті та перевірка	3	Дмитро Бондар	18.04.2026, 09:10:00	—	1 920 ₴

Рисунок 3.10 – Сторінка списку замовлень

3.3.3 Модуль позицій замовлення

3.3.3.1 Алгоритм додавання позиції до замовлення

У контексті відкритої картки замовлення користувач обирає тип позиції; за підказками у формі вказує позицію та кількість, формується POST-запит до відповідного ресурсу; після успішної відповіді списки позицій перезавантажуються через `read-only` подання.

Послідовність показано на рисунку 3.11.

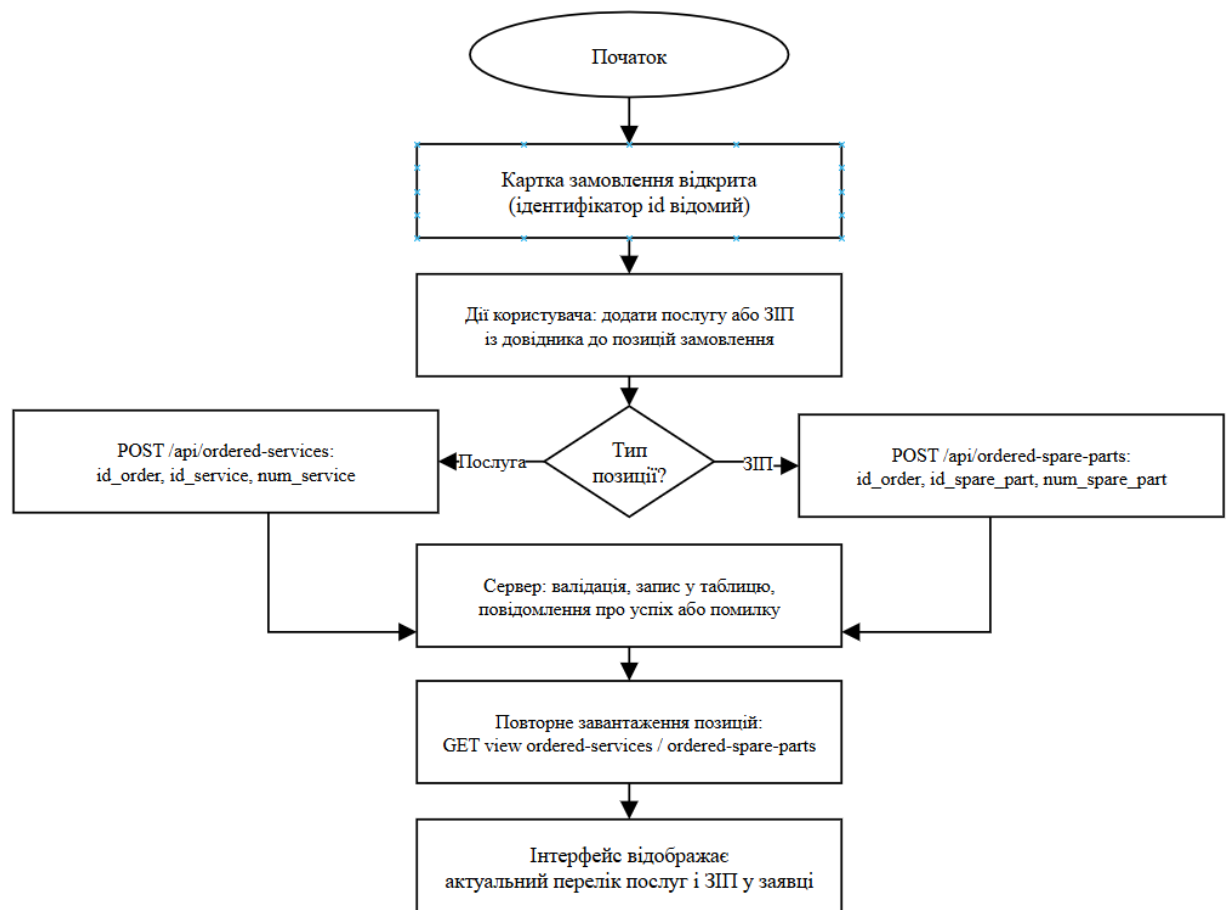


Рисунок 3.11 – Алгоритм додавання позиції (послуга або ЗІП) до замовлення

3.3.3.2 Функціональність модуля

Модуль забезпечує наступні функції:

- додавання, зміна та видалення рядків «замовлена послуга»;
- додавання, зміна та видалення рядків «замовлена запасна частина»;
- друк квитанції.

3.3.3.3 Реалізація та використані технології

Для позицій замовлення створено окремі API-ресурси `/api/ordered-services` та `/api/ordered-spare-parts` з узагальненими CRUD-операціями. Перегляд підготовлених списків виконується через `/api/ordered-services-view`

та `/api/ordered-spare-parts-view`, щоб картка замовлення отримувала уже зручні для відображення дані.

На клієнтській частині модуль реалізовано сторінкою `OrderDetails`. Інтерфейс сторінки наведено на рисунку 3.12.

Сервісний центр

- Замовлення
- Клієнти
- Працівники
- Послуги
- Запчастини
- Статистика
- Управління користувачами

Деталі замовлення № 11

← До списку Друк квитанції

Перегляд замовлення, послуг та запчастин

ID ЗАМ.	ID КЛ.	КЛІЄНТ	ТЕЛЕФОН	ПРИСТРІЙ	ОПИС	ID М.	МАЙСТЕР	СТВОРЕНО	ОРІЄНТ. ЗАВЕРШЕННЯ	ОРІЄНТ. ВАРТІСТЬ	З/П
11	1	Андрій Шевченко	+380971000001	ПК Intel Core i5	Видалення шкідливого ПЗ та налаштування драйверів.	1	Олексій Коваленко	06.05.2026, 16:10:00	—	800 ₴	0,11

Замовлені послуги Додати

НАЗВА ПОСЛУГИ	ТРИВАЛІСТЬ (ХВ)	ЦІНА (₴)	К-СТЬ	ДІЇ
Встановлення драйверів і базових програм	60	350 ₴	1	
Видалення вірусів та небажаного ПЗ	90	500 ₴	1	

Замовлені запчастини Додати

НАЗВА ЗАПЧАСТИНИ	ЦІНА (₴)	К-СТЬ	ДІЇ
USB Type-C кабель 1 м	150 ₴	1	

Вийти

Рисунок 3.12 – Картки замовлення: таблиці послуг і запасних частин

3.3.4 Модуль довідників послуг і запасних частин

3.3.4.1 Функціональність модуля

Модуль забезпечує наступні функції:

- ведення довідника послуг (назва, ціна, опис тощо за структурою сутності);
- ведення довідника запасних частин (облік одиниці, ціни, наявності);
- пошук і сортування записів у табличних списках довідників;
- повний CRUD через API з перевіркою унікальності та обробкою конфліктів при порушенні обмежень бази.

3.3.4.2 Реалізація та використані технології

Довідники послуг і запасних частин обслуговуються ресурсами /api/services та /api/spare-parts. Для перевірки даних застосовано Zod-схеми servicesZodSchema та sparePartsZodSchema.

На клієнті модуль представлено сторінками ServicesPage (рис. 3.13) і SparePartsPage (рис. 3.14).

Сервісний центр

Послуги
Довідник послуг сервісного центру

Пошук... Усі поля

ID ПОСЛ.	НАЗВА ПОСЛУГИ	ТРИВАЛІСТЬ (ХВ)	ЦІНА (₾)	ДІЇ
1	Діагностика комп'ютера або ноутбука	60	300 ₾	
2	Чищення системного блока від пилу	45	350 ₾	
3	Заміна термопасты	40	300 ₾	
4	Встановлення та налаштування Windows	120	600 ₾	
5	Встановлення драйверів і базових програм	60	350 ₾	
6	Видалення вірусів та небажаного ПЗ	90	500 ₾	
7	Заміна накопичувача HDD/SSD	60	450 ₾	
8	Відновлення даних з накопичувача	180	900 ₾	

Вийти

Рисунок 3.13 – Сторінка довідника послуг

Сервісний центр

Запчастини
Довідник запчастин та залишки на складі

Пошук... Усі поля

ID З/Ч	НАЗВА ЗАПЧАСТИНИ	ЦІНА (₾)	КІЛЬКІСТЬ НА СКЛАДІ	ДІЇ
1	Kingston Fury Beast DDR4 8GB	950 ₾	6	
2	Kingston Fury Beast DDR4 16GB	1 750 ₾	4	
3	Samsung 970 EVO Plus NVMe SSD 500GB	2 950 ₾	3	
4	Crucial MX500 SATA SSD 1TB	2 550 ₾	2	
5	Seagate Barracuda HDD 2TB	2 100 ₾	4	
6	Arctic MX-4 термопаста	220 ₾	12	
7	Noctua NT-H1 термопаста	310 ₾	5	
8	Cooler Master Hyper 212 Black Edition	1 450 ₾	2	

Вийти

Рисунок 3.14 – Сторінка довідника запасних частин

3.3.5 Модуль обліку клієнтів і персоналу

3.3.5.1 Функціональність модуля

Модуль забезпечує наступні функції:

- перегляд табличних списків клієнтів і співробітників;
- створення, редагування та видалення записів довідників через REST API з валідацією полів.

3.3.5.2 Реалізація та використані технології

Для довідників клієнтів і працівників використано спільний підхід до побудови CRUD-маршрутів: `/api/clients` та `/api/employees` формуються через `createCrudRouter` на основі `BaseService` і `Zod`-схем сутностей. Це дає змогу не дублювати типові операції створення, редагування, читання та видалення в різних частинах серверного коду.

На клієнті модуль представлено сторінками `ClientsPage` (рис. 3.15) і `EmployeesPage` (рис. 3.16).

ID КЛ.	КЛІЄНТ	ТЕЛЕФОН	ВСЬОГО ВИТРАЧЕНО	Є ЗНИЖКА	Дії
5	Віктор Поліщук	+380971000005	0 ₴	Ні	✎ ✖
4	Наталія Іваненко	+380971000004	3 600 ₴	Так	✎ ✖
10	Ілона Климчук	+380971000010	0 ₴	Ні	✎ ✖
6	Марія Романюк	+380971000006	2 980 ₴	Так	✎ ✖
2	Олена Кравчук	+380971000002	1 250 ₴	Ні	✎ ✖
7	Богдан Ткаченко	+380971000007	0 ₴	Ні	✎ ✖
1	Андрій Шевченко	+380971000001	2 920 ₴	Так	✎ ✖
8	Катерина Лисенко	+380971000008	2 850 ₴	Так	✎ ✖

Рисунок 3.15 – Сторінка обліку клієнтів

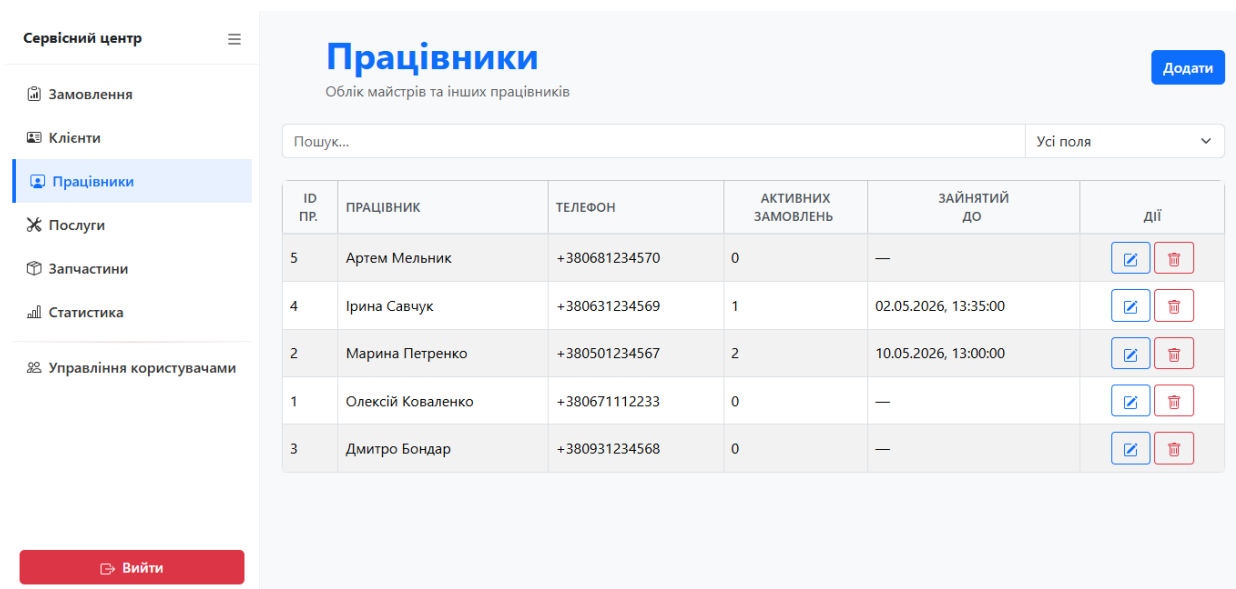


Рисунок 3.16 – Сторінка обліку працівників (майстрів)

3.3.6 Модуль подань і статистики

3.3.6.1 Функціональність модуля

Модуль забезпечує наступні функції:

- перегляд зведених даних по клієнтах, замовленнях, майстрах, позиціях у вигляді read-only API з можливістю пошуку та сортування в інтерфейсі;
- отримання місячної статистики замовлень за ключем дати (ендпоінт з первинним ключем month_start_date у вигляді рядка YYYY-MM-DD);
- відображення показників на сторінці статистики в інтерфейсі користувача.

3.3.6.2 Реалізація та використані технології

На сервері read-only подання обслуговуються через ReadOnlyBaseService, тобто для них передбачено лише операції читання. До цього блоку належать маршрути /api/clients-view, /api/orders-view,

/api/employers-view, /api/ordered-services-view, /api/ordered-spare-parts-view та /api/monthly-orders-statistics.

Відображення статистики й підсумкових таблиць виконується на сторінці StatisticsPage (рис. 3.17).

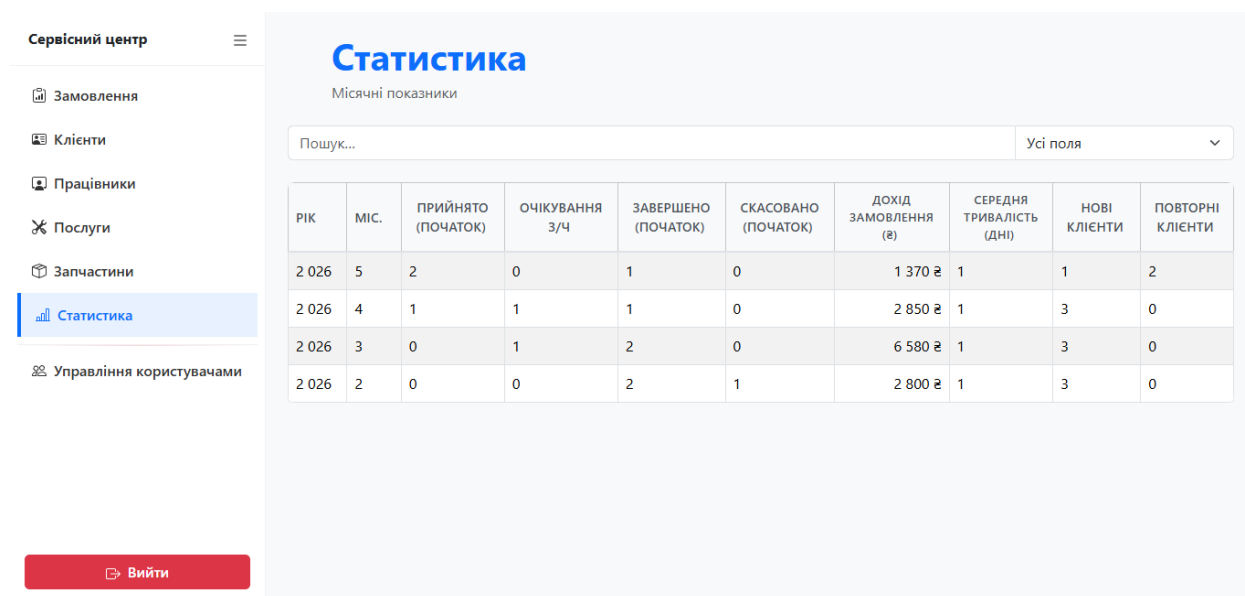


Рисунок 3.17 – Сторінка статистики та звітності

3.3.7 Модуль інфраструктури вебзастосунку

3.3.7.1 Функціональність модуля

Модуль забезпечує наступні функції:

- захист маршрутів: автентифікація та авторизація;
- єдиний HTTP-клієнт axios із базовою адресою API, перехопленнями запиту й відповіді;
- серверне ядро: прикладний шар Express, глобальні обробники помилок, CORS з підтримкою credentials, журналювання запитів;
- узагальнені сервіси BaseService та ReadOnlyBaseService для скорочення дублювання коду CRUD;
- планові завдання node-cron та конфігурація через змінні середовища (.env).

3.3.7.2 Реалізація та використані технології

Репозиторій організовано як npm workspaces: кореневий package.json об'єднує пакети client і server. Клієнтська частина використовує Vite, React 19, react-router-dom, react-bootstrap та axios, а серверна – Express 5, TypeORM 0.3, pg, zod, jsonwebtoken, cookie-parser та інші службові бібліотеки. Збірка клієнта виконується засобами Vite.

3.3.8 Проєктування бази даних

3.3.8.1 Структура бази даних

Проєктування бази даних спирається на вимоги до обліку сервісного центру: ідентифікація контрагентів, ведення замовлень із життєвим циклом, зв'язування замовлень із послугами та запасними частинами з контролем кількості, окремий облік користувачів вебзастосунку та механізму оновлення сесії. У таблицях використовується сурогатний ключ id для сумісності з ORM та ключі для забезпечення зв'язків і унікальності.

Агреговані подання доповнюють фізичну модель: вони інкапсулюють об'єднання таблиць, обчислення сум і показників і використовуються сервером API у режимі лише читання.

3.3.8.2 Взаємозв'язки між таблицями

Взаємозв'язки між таблицями реалізовано через наступні зовнішні ключі:

- auth_refresh_sessions.user_id посилається на users.id із правилом ON DELETE CASCADE – при видаленні користувача його refresh-сесії автоматично вилучаються;
- orders.id_client посилається на clients.id_client – кожне замовлення належить рівно одному клієнту (зв'язок «один до багатьох» з боку клієнта);

- `orders.id_emp` посилається на `employees.id_emp` – майстер може бути не призначений (поле допускає NULL), але якщо призначений, посилання має відповідати існуючому працівнику;
- `ordered_services` поєднує `orders` і `services`: поля `id_order` та `id_service` є зовнішніми ключами відповідно до `orders.id_order` та `services.id_service`; комбінація (`id_order`, `id_service`) унікальна, що відповідає правилу «в одному замовленні одна й та сама послуга не дублюється»;
- `ordered_spare_parts` аналогічно поєднує `orders` і `spare_parts` з унікальністю пари (`id_order`, `id_spare_part`).

Графічне узагальнення взаємозв'язків наведено на рисунку 3.18.

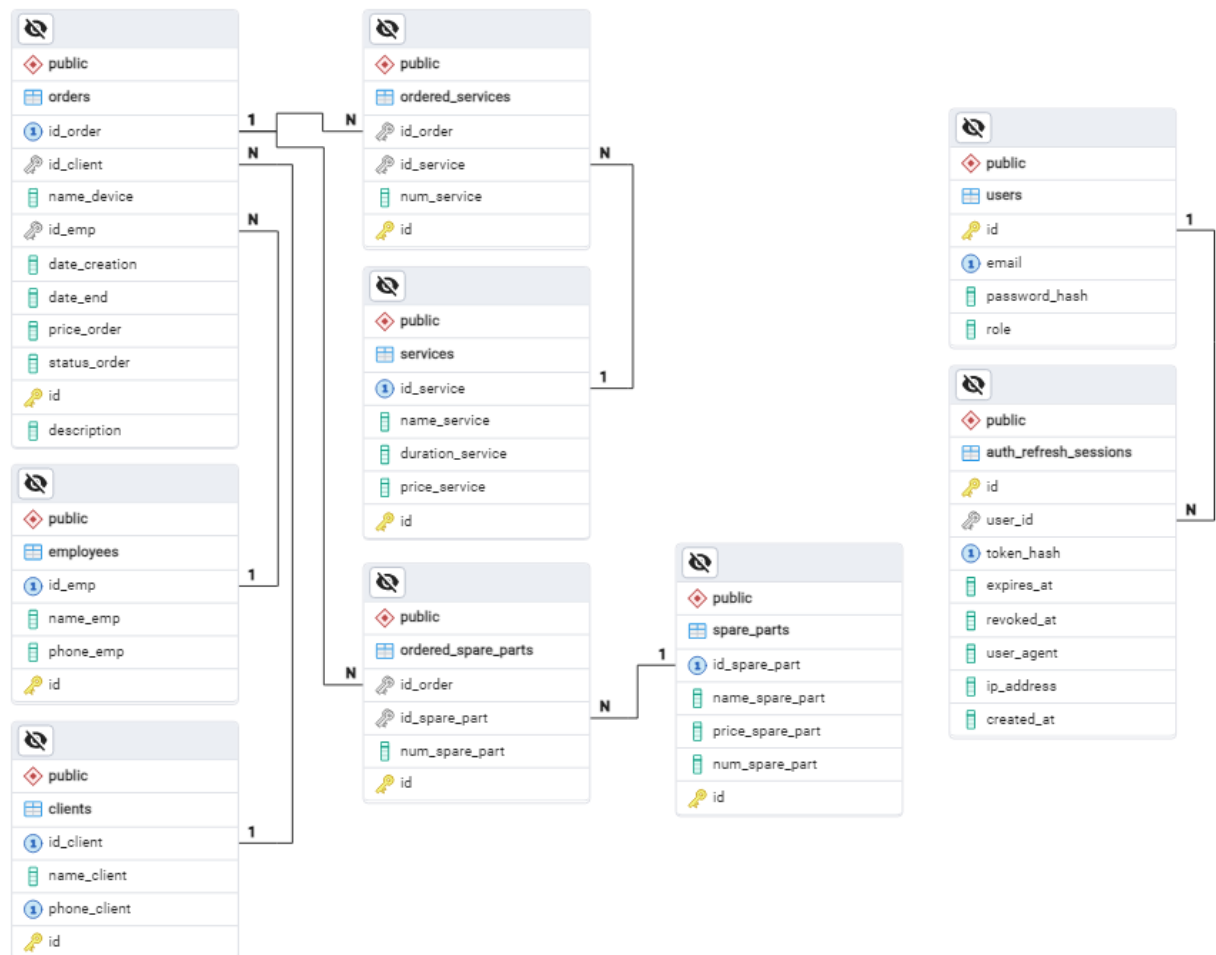


Рисунок 3.18 – Схема взаємозв'язків між таблицями

3.3.8.3 Індекссація та оптимізація

Для оптимізації запитів до бази даних використовуються такі індекси:

- індекс на `auth_refresh_sessions.user_id` для швидкого пошуку активних refresh-сесій за користувачем;
- індекс на `auth_refresh_sessions.expires_at` для вибірок за терміном дії токена та очищення прострочених записів;
- індекс на `auth_refresh_sessions.revoked_at` для обліку відкликаних сесій і селективного обслуговування журналу сесій;
- індекс на `orders.id_client` для швидкого об'єднання з довідником клієнтів і вибірки замовлень за контрагентом;
- індекс на `orders.id_emp` для швидкого фільтрування замовлень за призначеним майстром;
- індекс на `orders.date_creation` у порядку спадання для швидкого сортування списків замовлень, подань та місячних агрегатів за часом створення заявки;
- частковий складений індекс на `orders` за полями `id_emp` та `date_creation` для рядків зі статусом «Прийнято» та непустим `id_emp` – для прискорення чергової логіки оцінки термінів обслуговування одного майстра.

Окремо первинні ключі та обмеження унікальності таблиць автоматично підтримуються індексами B-tree у PostgreSQL. За зростанням обсягів даних доцільно періодично виконувати ANALYZE для оновлення статистики планувальника й за потреби перевіряти план виконання критичних запитів командами EXPLAIN.

3.4 Висновки за розділом 3

У третьому розділі наведено розробку вебзастосунку «Сервісний центр» для обліку роботи комп'ютерного сервісу. Описано взаємодію користувача з інтерфейсом, обробку запитів на сервері та збереження даних у

централізованій PostgreSQL-базі замість розрізнених файлів чи локальних копій. Окремо подано сценарії роботи, які показують дії користувачів, порядок збереження замовлення та реакцію системи на типові помилки.

Архітектуру побудовано за схемою тонкого клієнта: у браузері працює React-додаток, серверна частина реалізована на Node.js і Express, а дані зберігаються в PostgreSQL. Такий розподіл зручний для майстерні, оскільки перевірка прав, форматів даних і бізнес-правил зосереджена в одному місці. Автентифікацію організовано через access token і refresh-сесію, вхідні дані перевіряються за правилами Zod, а робота з таблицями виконується через TypeORM.

У підрозділах розглянуто функціональні схеми, потік HTTP-запитів і діаграми варіантів використання адміністратора та користувача. Для ключових процесів – входу в систему, створення й супроводу замовлення та роботи з його позиціями – наведено блок-схеми алгоритмів.

Описано сім логічних модулів: автентифікація та керування користувачами, облік клієнтів і персоналу, замовлення, довідники послуг і запасних частин, позиції замовлення, подання й статистика, а також інфраструктура вебзастосунку. Разом вони покривають основний цикл роботи сервісу: від входу користувача і заведення клієнта до формування заявки, додавання робіт або ЗІП, перегляду підсумків і підтримки технічного каркаса системи.

Окремо розкрито проектування бази даних у PostgreSQL: таблиці предметної області, облік користувачів і refresh-сесій, подання для екранів та статистики, зовнішні ключі й індекси на полях, які найчастіше використовуються у запитах.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосування програми

4.1.1 Призначення та область застосування

Вебзастосунок «Сервісний центр» задуманий як робоче «вікно» до єдиної бази даних для невеликої сервісної майстерні: замість розкиданих таблиць у файлах або нотаток на папері в одному місці збираються клієнти, майстри, заявки, перелік послуг і запчастин, історія виконання. Це не окремий магазин для клієнтів у відкритому доступі, а внутрішній інструмент персоналу: прийом замовлення, призначення виконавця, зміна статусу, доповнення рядками послуг чи складових – усе це лишається в межах організації, що розгорнула сервер і видала облікові записи.

Область застосування – облік і супровід ремонту й обслуговування комп'ютерної техніки там, де вже є локальна мережа (або інший стабільний канал) до вузла з Node.js і PostgreSQL. Типовий сценарій – одна точка або кілька робочих місць у тій самій структурі; система орієнтована на десятки одночасних сесій у межах навчально-демонстраційного та малого виробничого навантаження, а не на публічний інтернет-сервіс з мільйонами відвідувань.

Користувачі за призначенням діляться на дві ролі, і це варто тримати в голові ще до читання інструкції з установлення. Адміністратор – той, хто заводить нові облікові записи, бачить перелік користувачів і може прибрати запис, якщо політика безпеки це дозволяє; без цієї ролі «роздати» логіни колегам просто нікому. Користувач (умовний майстер або адміністратор пункту прийому – як домовляться на місці) працює з предметною частиною: довідники, замовлення, рядки замовлення, подання для перегляду зведень. Така поділка не дробить інтерфейс за зайвим: радше нагадує, що доступ до даних має бути не «на всіх під один пароль», а з зрозумілим розмежуванням відповідальності.

4.1.2 Методи реалізації та класи вирішуваних завдань

З погляду техніки застосовано поширену для вебпроектів схему «тонкий клієнт»: у браузері працює інтерфейс на React, на сервері – шар HTTP API на Express, дані зберігаються в реляційній базі PostgreSQL, а зв'язок із таблицями організовано через TypeORM. Обмін між клієнтом і сервером ведеться за протоколом HTTP або HTTPS, тіла запитів і відповідей у форматі JSON; для захисту операцій використовують короткоживучий маркер доступу (JWT) і окремий механізм оновлення сесії з журналом на сервері. Вхідні поля перевіряють схемами Zod, паролі в базі тримають у вигляді криптографічних гешів – це вже не «фішка», а стандартний мінімум, але саме він дозволяє не переписувати бізнес-логіку кожного разу, коли хтось вводить некоректну дату або забуває поле.

До класів задач, які така архітектура перекриває, належать: ведення довідників клієнтів і співробітників; повний життєвий цикл замовлення (фіксація техніки, статусів і дат, прив'язка до клієнта й виконавця); підтримка номенклатури послуг і запасних частин; формування рядків замовлених послуг та рядків замовлених запчастин без дублювання тієї самої позиції; робота з поданнями та місячною статистикою, щоб не тягнути в клієнт складні агрегації; автентифікація й розмежування прав залежно від ролі. Окремим шаром лишаються інфраструктурні речі – маршрутизація, спільні CRUD-сервіси, прибирання прострочених refresh-сесій – без них код головних екранів швидко перетворився б на хаотичні повтори.

4.1.3 Обмеження застосування

Програма розрахована на роботу з сервером у режимі реального часу: якщо зникає мережа або падає база, інтерфейс може показати помилку й не дасть надійно зберегти зміни – офлайн-режим із подальшою синхронізацією для дипломного варіанта не пропонується. Не передбачено окремих

мобільних застосунків під Android чи iOS: зручність на смартфоні залежить від ширини екрана й браузера, але основна верстка орієнтована на робочий стіл або ноутбук. Експорт у зовнішні бухгалтерські системи, електронний документообіг і публічний запис клієнтів «в чергу через сайт» у поточну версію також не входять – їх можна планувати як розвиток (див. п. 4.6), а не як обов’язкову властивість. Нарешті, безпека залежить від того, хто адмініструє сервер: слабкі паролі, відкритий без TLS доступ із публічної мережі або застаріла СУБД – це вже питання експлуатації, які сама програма не «вилікує».

4.1.4 Мінімальна конфігурація технічних засобів

Вимоги до робочого місця користувача:

- персональний комп’ютер або ноутбук із підключенням до мережі;
- операційна система: сучасна ОС сімейства Windows / Linux / macOS;
- веббраузер останніх стабільних версій (наприклад, на базі Chromium, Firefox, Safari);
- оперативна пам’ять: не менше 4 Гб для комфортної роботи в браузері з іншими застосунками;
- роздільна здатність екрана: не менше 1280×720 для коректного відображення інтерфейсу.

Вимоги до сервера та бази даних:

- процесор: x86-64, 2 ядра і більше;
- оперативна пам’ять: не менше 4–8 Гб залежно від обсягу даних і числа одночасних користувачів;
- дисковий простір: з урахуванням обсягу бази даних та журналів – не менше 20 Гб вільного місця для базової експлуатації;
- мережева взаємодія: порт для прослуховування HTTP/HTTPS API сервером застосунку; доступ клієнтів до цього вузла в мережі.

4.2 Інструкція по роботі з програмою

4.2.1 Встановлення програми

Встановлення вебзастосунку полягає у розміщенні на сервері клієнтської частини, серверного API та бази даних PostgreSQL. На вузлі розгортання попередньо мають бути встановлені Node.js, npm і PostgreSQL; також слід передбачити мережевий доступ до вебінтерфейсу та API лише з тих робочих місць, для яких система призначена.

Код проєкту отримують із репозиторію або архіву та завантажують на сервер у робочий каталог. У корені проєкту виконують «npm install», після чого команда «npm run build» формує зведений каталог «dist»: у «dist/client» розміщується зібраний React/Vite-інтерфейс, а в «dist/server» – серверна частина Express. Клієнтські файли можна публікувати через вебсервер, а серверну частину запускати як Node.js-процес.

Окремо налаштовують PostgreSQL: створюють базу даних, користувача з потрібними правами та імпортують SQL-дамп зі структурою таблиць і початковими даними. У каталозі серверної частини розміщують файл «.env», де задають параметри підключення до БД, секрети JWT і refresh-механізму, режим роботи та інші змінні середовища. Адресу API у клієнтській конфігурації задають як мережеву адресу сервера.

Після збірки й налаштування серверний застосунок запускають, наприклад командою «node index.js» із каталогу «dist/server» або через менеджер процесів.

Узагальнений порядок дій показано на рисунку 4.1.

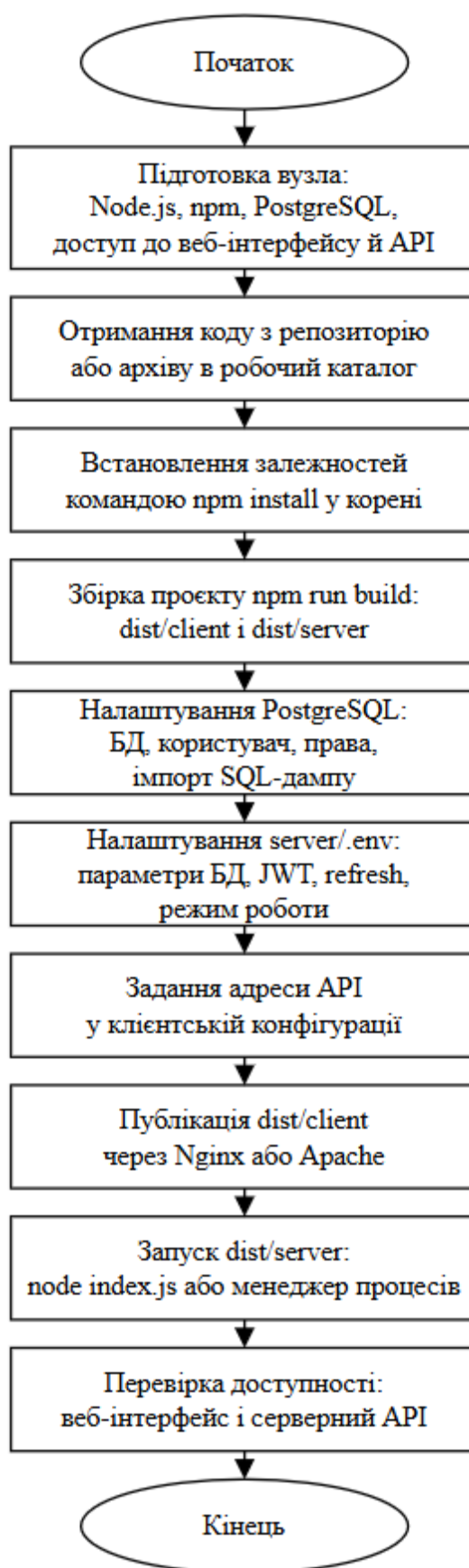


Рисунок 4.1 – Схема процесу встановлення застосунку

4.2.2 Запуск програми

Після розгортання оператор відкриває браузер і переходить за URL вебзастосунку: це може бути локальна IP-адреса, ім'я хоста або домен організації. Першою відкривається сторінка входу, вигляд якої відповідає рисунку 3.7. Користувач вводить електронну пошту та пароль, після чого система відкриває захищені розділи відповідно до його ролі.

Якщо система запускається вперше і в базі ще немає користувачів, перед входом потрібно створити обліковий запис адміністратора. Для цього використовується службова утиліта «add-user» у пакеті «server»; після створення адміністратора вже через вебінтерфейс можна додавати інших користувачів і призначати їм ролі.

4.2.3 Методика роботи з програмою

Щоденна робота будується навколо замовлення. Спочатку варто заповнити довідники послуг (рис. 3.13), запчастин (рис. 3.14), клієнтів (рис. 3.15) та майстрів (рис. 3.16). Відповідно до алгоритму, наведеного на рисунку 3.9, оператор створює запис замовлення, відстежує статус і змінює поля через інтерфейс списку й картки (рис. 3.10, 3.12). Перелік виконаних послуг та використаних запчастин додають у контексті відкритої картки відповідно до алгоритму, наведеного на рисунку 3.11. Статистика та звітність доступні на окремій сторінці (рис. 3.17).

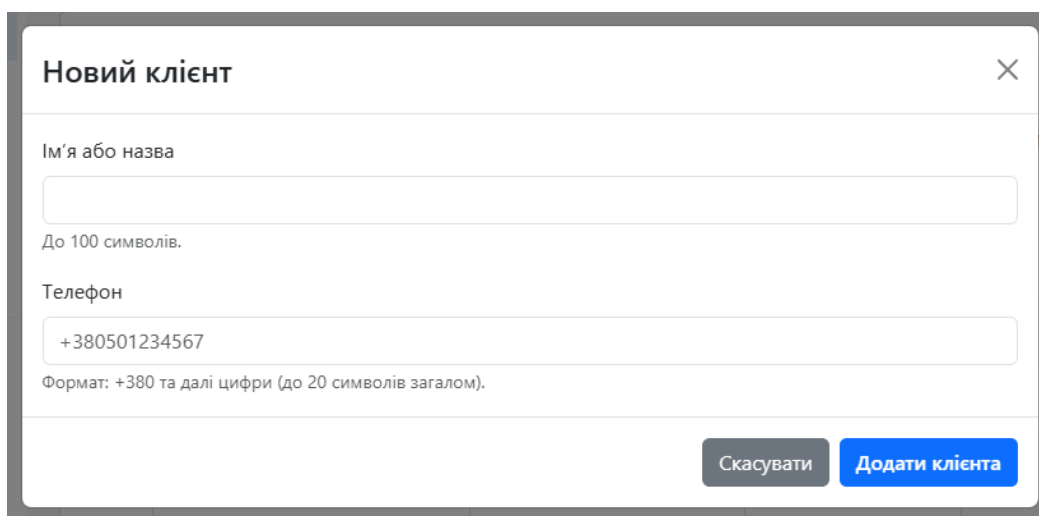
Паралельно з роботою над замовленнями адміністратор веде облік користувачів (рис. 3.8): додає та видаляє користувачів, призначає ролі. Ролі призначаються за регламентом організації, а технічні обмеження визначені на рівні сервера та бази даних.

4.3 Приклад роботи користувача з програмою

Розглянемо узагальнений сценарій роботи користувача, який показує шлях від входу в систему до створення заявки з позиціями та перевірки зведених даних. На початку зміни працівник відкриває вебінтерфейс після розгортання системи адміністратором і отримання власного облікового запису (див. п. 4.2). Перед оформленням замовлення він перевіряє наявність потрібного клієнта та відповідального майстра в довідниках, а за потреби додає відсутні записи (рис. 4.2).

Після цього користувач уточнює, чи внесено до довідників необхідні послуги та запасні частини. Далі він відкриває список замовлень, створює нову заявку (рис. 4.3) або продовжує роботу з чернеткою, заповнює клієнта, майстра, статус і опис техніки, після чого додає до замовлення послуги та, за потреби, позиції ЗІП та послуг (рис. 4.4).

Окремо передбачено адміністративний мікросценарій: адміністратор створює обліковий запис для нового працівника та призначає йому роль користувача. Таким чином система охоплює не лише оформлення заявок, а й керування доступом до службових даних.



Новий клієнт

Ім'я або назва

До 100 символів.

Телефон

+380501234567

Формат: +380 та далі цифри (до 20 символів загалом).

Скасувати Додати клієнта

Рисунок 4.2 – Додавання нового клієнта

Нове замовлення

Клієнт * Працівник Статус

Пошук за будь-яким полем кє Пошук за будь-яким полем кє Прийнято

Пристрій *

Опис

Необов'язково, до 1000 символів

0 / 1000

Вартість (₴)

0

Скасувати Створити замовлення

Рисунок 4.3 – Додавання нового замовлення

Додати послугу

Послуга *

Пошук послуги...

Кількість

1

Скасувати Додати

Рисунок 4.4 – Формування замовлення: додавання послуг і запасних частин

4.4 Тестування роботи програми

4.4.1 Функціональне тестування

Функціональне тестування виконувалось за сценаріями, що відповідають основним діям користувача у вебзастосунку. Оскільки промислове тестування не проводилося, результати подано як навчальну перевірку на контрольному наборі даних.

Методи тестування:

а) модульне тестування:

- 1) перевірка валідації введених даних;
- 2) перевірка окремих сервісних операцій;
- 3) перевірка обробки типових помилок;

б) інтеграційне тестування:

- 1) перевірка взаємодії клієнта із сервером;
- 2) перевірка збереження даних у PostgreSQL;
- 3) перевірка отримання відповідей API;

в) інтерфейсне тестування:

- 1) перевірка входу до системи;
- 2) перевірка навігації між сторінками;
- 3) перевірка відображення списків, форм і повідомлень.

Результати функціонального тестування наведено в таблиці 4.1.

Таблиця 4.1 – Результати функціонального тестування

Функціонал	Тест-кейси	Результат	Виправлені помилки
Вхід і вихід із системи	6	Успішно	1
Робота з довідниками	8	Успішно	1
Створення та редагування замовлення	9	Успішно	2
Додавання послуг і запасних частин	6	Успішно	1
Перегляд статистики	4	Успішно	0

Аналіз функціонального тестування показав, що основні сценарії роботи програми виконуються коректно.

Результати тестування (рис. 4.5):

- загальна кількість виконаних тестів: 33;
- успішно пройдено: 28;
- виявлено помилок: 5;
- виправлено помилок: 5.

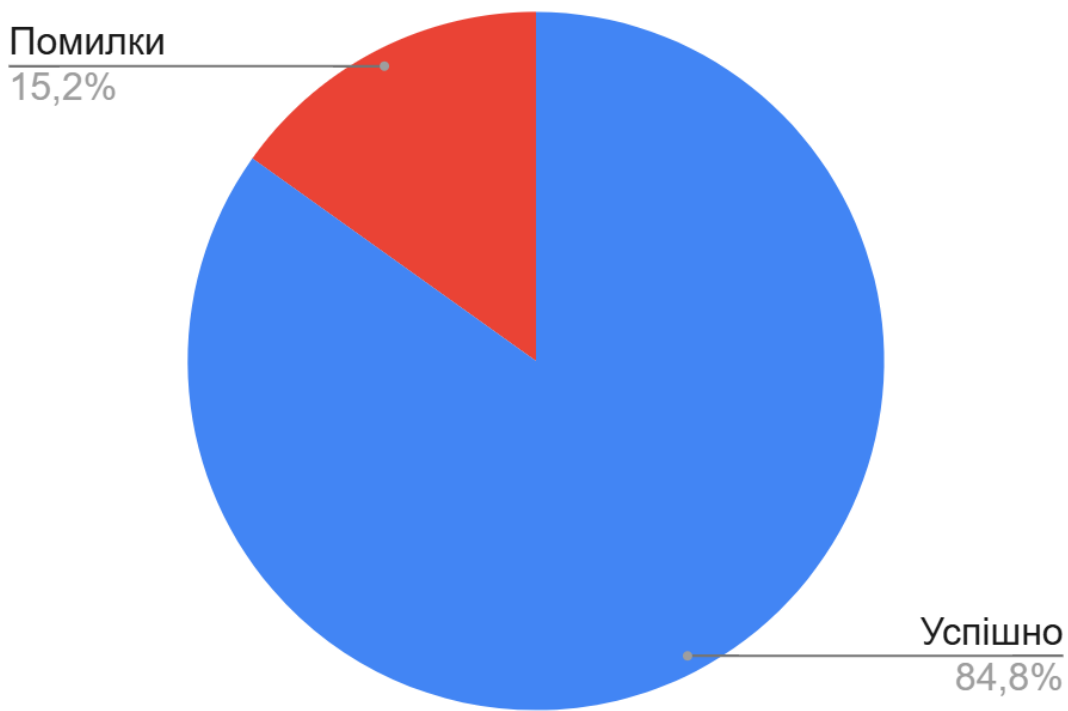


Рисунок 4.5 – Статистика успішності тестування модулів системи

Виявлені помилки переважно були пов’язані з повідомленнями інтерфейсу, перевіркою форм і оновленням списків після збереження. Виявлені помилки наведено в таблиці 4.2.

Таблиця 4.2 – Аналіз виявлених помилок

Тип помилки	Пріоритет	Час виявлення	Час виправлення	Статус
Повідомлення про помилку входу	Середній	02.05.2026	02.05.2026	Виправлено
Оновлення списку замовлень після збереження	Середній	03.05.2026	04.05.2026	Виправлено
Перевірка обов’язкових полів форми	Високий	04.05.2026	04.05.2026	Виправлено
Відображення довгих назв у таблиці	Низький	05.05.2026	06.05.2026	Виправлено

4.4.2 Тестування продуктивності

Тестування продуктивності мало орієнтовний характер. Перевірялося, як швидко система відповідає на типові запити при невеликому навантаженні, характерному для малої сервісної майстерні. Результати тестування продуктивності наведено в таблиці 4.3.

Таблиця 4.3 – Показники часу відповіді API на навчальному стенді

Операція	Кількість умовних користувачів	Середній час відповіді, мс	Результат
Відкриття списку замовлень	5	42	Успішно
Відкриття списку замовлень	10	58	Успішно
Відкриття списку замовлень	20	96	Успішно
Збереження картки замовлення	10	64	Успішно
Перегляд статистики	10	118	Успішно

Отримані дані (рис. 4.6) показують, що для умов невеликої майстерні швидкодія системи є достатньою: основні операції виконуються менше ніж за одну секунду. Найбільший час відповіді спостерігається при перегляді статистики, оскільки в цьому випадку використовуються зведені дані.

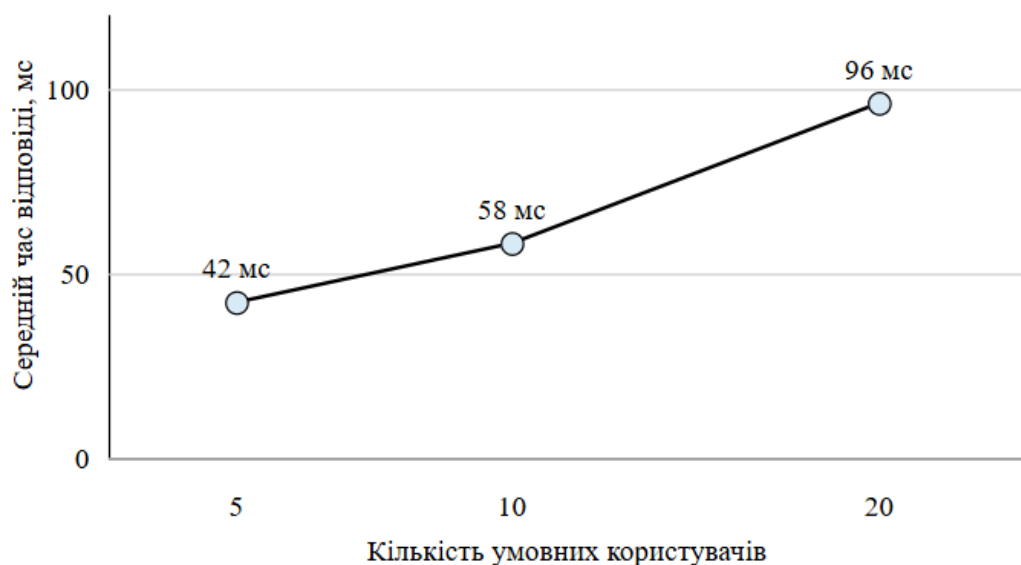


Рисунок 4.6 – Залежність часу відповіді від кількості користувачів

4.4.3 Тестування безпеки

Тестування безпеки виконувалося за внутрішнім чек-листом з урахуванням типових підходів до перевірки вебзастосунків [20]. Результати тестування безпеки наведено в таблиці 4.4.

Таблиця 4.4 – Результати тестування безпеки

Тип перевірки	Кількість перевірок	Виявлені проблеми	Усунені проблеми
Автентифікація	6	1	1
Ролі доступу	5	1	1
Refresh-сесії	4	1	1
Валідація даних	5	0	0

Проведені перевірки показали, що базові засоби захисту реалізовано коректно: доступ до закритих маршрутів контролюється, ролі користувачів враховуються, а сесії можуть бути відкликані. Виявлені зауваження стосувалися повідомлень інтерфейсу та організаційних умов експлуатації.

4.4.4 Аналіз користувацького досвіду

Проведено експертну оцінку серед п'яти користувачів. Кожен оцінив інтерфейс за п'ятибальною шкалою за п'ятьма критеріями. Усереднені результати наведено у таблиці 4.5 та на рисунку 4.7.

Таблиця 4.5 – Результати опитування зручності вебінтерфейсу (1 – «важко», 5 – «зручно»)

Критерій	Середній бал
Зрозумілість входу та повідомлень про помилку	4,4
Навігація між довідниками й замовленнями	4,0
Швидкість відгуку при збереженні картки	4,2
Читабельність таблиць на ноутбукі 1366×768	3,6
Зрозумілість відображення статистичних даних	4,3

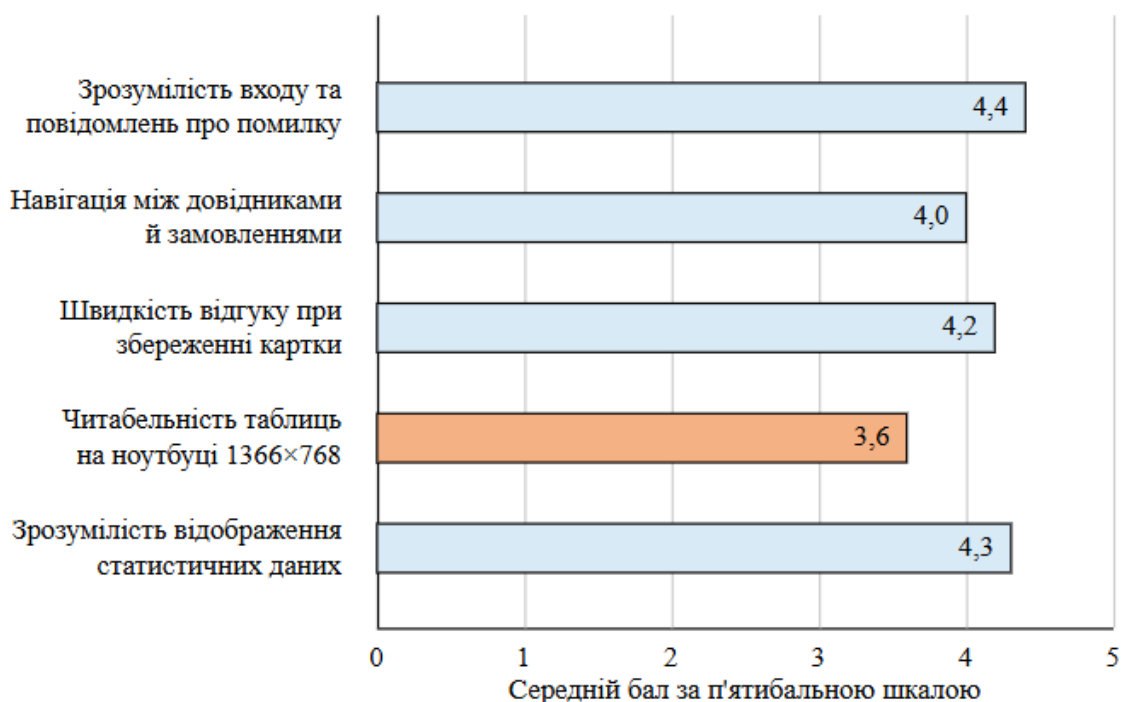


Рисунок 4.7 – Середні оцінки зручності за критеріями

4.5 Рекомендації щодо використання

З урахуванням результатів перевірок, наведених у п. 4.4, та практичного сценарію роботи з п. 4.3 доцільно визначити основні рекомендації щодо подальшої експлуатації системи. Для забезпечення стабільної та безпечної роботи вебзастосунку слід використовувати зворотний проксі-сервер із підтримкою TLS, регулярно створювати резервні копії бази даних PostgreSQL та своєчасно оновлювати програмні залежності. Такі заходи зменшують ризик втрати даних, спрощують відновлення після технічних збоїв і підвищують загальну надійність системи.

Під час організації роботи користувачів необхідно підтримувати довідники клієнтів і майстрів в актуальному стані до початку активного ведення замовлень. Це дає змогу уникнути некоректних посилань у картках заявок і забезпечує коректність подальших звітів. Також слід дотримуватися принципу персональної відповідальності за облікові записи: використання спільного логіна ускладнює аналіз журналу сесій та розслідування можливих

інцидентів. У разі зміни статусу замовлення доцільно фіксувати коротке пояснення виконаних дій у полі опису або у внутрішньому регламенті майстерні, щоб зберігати зрозумілу історію ремонту.

Окрему увагу варто приділяти технічним аспектам щоденної роботи. Не рекомендується одночасно редагувати одне й те саме замовлення у кількох вкладках браузера без попереднього оновлення даних, оскільки це може призвести до перезапису змін. У разі появи повідомлення про мережеву помилку слід насамперед перевірити доступність сервера та правильність URL-адреси застосунку. Адміністратору системи необхідно періодично переглядати список облікових записів, видаляти неактуальні записи та контролювати, щоб тестові паролі або демонстраційні облікові дані не використовувалися в робочому середовищі.

4.6 Плани з розвитку

У найближчій перспективі доцільно оптимізувати сторінки застосунку під вузькі екрани малої ширини та додати темну тему сторінки.

У середньостроковій перспективі корисно розглянути сповіщення про зміну статусу замовлення, аудит-лог дій щодо критичних полів замовлення й розширення наявного пошуку складнішими фільтрами. Якщо сервіс виросте до кількох точок, знадобиться хоча б розділення по філіях або по напрямках обліку в одній базі.

Довгостроково продукт можна розвивати в бік глибшої інтеграції з бухгалтерськими системами або з постачальниками ЗІП через імпорт накладних, мобільного PWA для майстра «з планшета в залі», механізму гарантійних ремонтів та прогнозу навантаження на основі накопичених статусів.

ВИСНОВКИ

З метою впорядкування обліку в комп'ютерному сервісі в роботі вирішено практичне завдання створення вебзастосунку «Сервісний центр» для ведення клієнтів, працівників, замовлень, послуг, запасних частин і пов'язаних операцій. Продукт орієнтований на щоденний облік невеликої майстерні, який часто ведеться в таблицях, журналах або усних домовленостях.

Проведено аналіз предметної області та підходів до обліку роботи сервісного центру. Розглянуто хмарні сервіси, CRM-системи й табличний облік; встановлено, що готові рішення часто є надлишковими для малої майстерні, а таблиці не підтримують цілісні зв'язки між клієнтом, замовленням, послугами й запасними частинами. Це обґрунтувало доцільність власного вебзастосунку з централізованою базою даних.

Сформульовано функціональні та нефункціональні вимоги до програмного продукту. До основних функцій віднесено ведення довідників, роботу із замовленнями та їх позиціями, зведений перегляд через подання, автентифікацію і розмежування прав. Нефункціональні вимоги охопили продуктивність, обробку помилок, захист даних, супроводжуваність і можливість розширення.

Спроектовано архітектуру програмного комплексу за клієнт-серверною моделлю: React-інтерфейс працює у браузері, сервер реалізовано на Node.js та Express, а дані зберігаються в PostgreSQL. Перевірку прав, валідацію, обробку запитів і роботу з базою зосереджено на сервері. Для обміну використано HTTP/JSON, для доступу до даних – TypeORM, для перевірки вхідних даних – Zod.

Розроблено модель бази даних предметної області. Вона охоплює користувачів і refresh-сесії, клієнтів, співробітників, замовлення, послуги, запасні частини та рядки замовлених позицій. Для читання складених даних

передбачено SQL-подання, а для часто використовуваних полів визначено індекси.

Реалізовано основні модулі вебзастосунку: автентифікацію й керування користувачами, облік клієнтів і персоналу, замовлення, довідники, позиції замовлення, подання та статистику, а також інфраструктурний шар запуску й збірки. Передбачено вхід за електронною поштою та паролем, access-токен, refresh-механізм, серверний облік сесій і рольовий доступ до адміністративних функцій.

Підготовлено матеріали для експлуатації програмного продукту: описано призначення, область застосування, встановлення, запуск і методику роботи, наведено типовий сценарій оформлення замовлення. Для розгортання подано послідовність дій від підготовки вузла й налаштування PostgreSQL до збірки клієнта, запуску сервера та перевірки доступності.

Виконано перевірку працездатності системи на навчальному стенді. У межах функціонального тестування умовно перевірено 33 тест-кейси за основними сценаріями роботи; виявлені 5 помилок було усунуто до оформлення звіту. Орієнтовні виміри продуктивності показали час відкриття списку замовлень 42, 58 і 96 мс при 5, 10 і 20 умовних користувачах; перевірки безпеки підтвердили коректну роботу автентифікації, ролей доступу, refresh-сесій і перевірки введених даних.

Практична значущість результату полягає у використанні застосунку як навчального прототипу або основи для внутрішньої системи малої майстерні. Система зменшує залежність від розрізнених файлів, підтримує централізоване зберігання даних і робить стан замовлень видимим для персоналу. Подальший розвиток доцільно спрямувати на звіти, розширений пошук, журнал аудиту, сповіщення, підтримку кількох точок обслуговування та інтеграції.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. RO App. Repair Shop Software. URL: <https://remonline.app/repair-shop-software/> (date of access: 14.05.2026).
2. HubSpot. HubSpot CRM. URL: <https://www.hubspot.com/crm/> (date of access: 14.05.2026).
3. Google. Google Sheets Help. URL: <https://support.google.com/docs/topic/9054603> (date of access: 14.05.2026).
4. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> (date of access: 14.05.2026).
5. React. Documentation. URL: <https://react.dev/> (date of access: 14.05.2026).
6. React Router. Documentation. URL: <https://reactrouter.com/> (date of access: 14.05.2026).
7. Node.js. Documentation. URL: <https://nodejs.org/docs/latest/api/> (date of access: 14.05.2026).
8. Express. Documentation. URL: <https://expressjs.com/> (date of access: 14.05.2026).
9. Zod. Documentation. URL: <https://zod.dev/> (date of access: 14.05.2026).
10. Jones M. B. JSON Web Token (JWT) : RFC 7519 / M. B. Jones, J. Bradley, N. Sakimura. URL: <https://www.rfc-editor.org/rfc/rfc7519> (date of access: 14.05.2026).
11. TypeORM. Documentation. URL: <https://typeorm.io/> (date of access: 14.05.2026).
12. PostgreSQL. Documentation. URL: <https://www.postgresql.org/docs/> (date of access: 14.05.2026).
13. ECMA International. ECMAScript Language Specification. URL:

<https://tc39.es/ecma262/> (date of access: 14.05.2026).

14. Visual Studio Code. Documentation. URL: <https://code.visualstudio.com/docs> (date of access: 14.05.2026).

15. Vite. Guide. URL: <https://vite.dev/guide/> (date of access: 14.05.2026).

16. Bootstrap. Documentation. URL: <https://getbootstrap.com/docs/> (date of access: 14.05.2026).

17. bcrypt. npm package documentation. URL: <https://www.npmjs.com/package/bcrypt> (date of access: 14.05.2026).

18. OWASP Foundation. Web Security Testing Guide. URL: <https://owasp.org/www-project-web-security-testing-guide/> (date of access: 14.05.2026).

ДОДАТОК А
Технічне завдання

Вступ

Вебзастосунок «Сервісний центр» призначений для автоматизації обліку та супроводу діяльності сервісного центру з ремонту й обслуговування комп'ютерної техніки (прийом замовлень, робота з клієнтами й персоналом, облік послуг і запасних частин). Область застосування – внутрішні процеси сервісної майстерні або невеликої мережі сервісів, де потрібне централізоване ведення даних і обмежений доступ за ролями. Об'єктом використання є персональні комп'ютери та інші пристрої з сучасним веббраузером і доступом до мережі (локальної чи глобальної) для взаємодії з серверною частиною системи.

A.1 Підстави до розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу, видане 07 квітня 2026 року. Найменування теми розробки – «Інженерія програмного забезпечення комп'ютерного сервісу».

A.2 Призначення розробки

Вебзастосунок «Сервісний центр» призначений для автоматизації процесів обліку роботи комп'ютерного сервісу – від збереження відомостей про клієнтів і співробітників (майстрів) до оформлення замовлень на ремонт і відстеження статусів їх виконання, ведення номенклатури послуг і запасних частин, а також фіксації в замовленні переліку наданих послуг і зарезервованих чи витрачених комплектуючих. Для аналітики та зручного перегляду передбачено роботу з поданнями в базі даних; доступ до функцій системи забезпечується після автентифікації користувача й розмежування повноважень між адміністратором та користувачем застосунку, зокрема для

централізованого створення облікових записів, перегляду їх списку та видалення з дотриманням встановлених обмежень.

Його експлуатаційне призначення полягає в полегшенні щоденної роботи персоналу сервісу: облік ведеться через браузер як єдине «вікно» до даних без розпорошення інформації по окремих файлах, а надійність і узгодженість записів підтримуються завдяки серверній обробці запитів і централізованому зберіганню даних.

А.3 Вимоги до програми чи програмного виробу

А.3.1 Вимоги до функціональних характеристик

Користувач з відповідними правами має можливість створювати, переглядати, оновлювати й видаляти записи в базі даних для сутностей, що відповідають предметній області: клієнти, співробітники, замовлення, послуги, запасні частини, рядки замовлених послуг, рядки замовлених запасних частин. Для замовлень передбачається підтримка життєвого циклу (наприклад, статуси виконання), узгоджена з моделлю даних на сервері, а також друк квитанції зі сторінки деталей замовлення.

Система має надавати доступ до зведеної інформації для зручного перегляду клієнтів, замовлень, співробітників, замовлених послуг і замовлених запасних частин. Оновлення цих даних виконується через основні записи предметної області.

Реєстрація нових облікових записів у системі виконується лише адміністратором. Вхід у систему – за електронною поштою та паролем. Після успішної автентифікації клієнтська частина отримує маркер доступу; для продовження сесії використовується механізм оновлення сесії із використанням захищених параметрів зберігання в браузері. Доступ до адміністративних функцій (управління користувачами) надається лише ролі адміністратора; відмова в доступі має супроводжуватись зрозумілим повідомленням для користувача.

Вхідні дані запитів проходять серверну валідацію; у разі некоректних даних повертається структурована відповідь про помилку. Взаємодія клієнта й сервера має виконуватись через мережевий інтерфейс у погодженому структурованому форматі.

Результатами роботи є відображення списків і карток записів у вебінтерфейсі, пошук і сортування табличних даних, підказки при заповненні форм, повідомлення про успішні операції або помилки та друк квитанції замовлення; за потреби подальшими версіями можуть бути додані експорт звітів.

У штатному режимі при локальній мережі або стабільному підключенні до сервера основні операції перегляду та збереження записів мають виконуватись без помітних для користувача затримок (порядку секунд) за умови відповідності параметрів технічних засобів і навантаження.

A.3.2 Вимоги до надійності

Система має забезпечувати стійке функціонування в межах обраного програмно-апаратного середовища: контроль цілісності та коректності вхідних даних на рівні валідації; обробку типових помилок збереження (наприклад, конфлікт унікальності обмежень у базі даних) з повідомленням користувачу без аварійного завершення роботи сервера.

Захист від несанкціонованого доступу до даних забезпечується обов'язковою автентифікацією для бізнес-операцій, перевіркою маркера доступу на кожному захищеному запиті та розмежуванням прав за ролями на сервері.

Сесії оновлення маркера доступу мають вести облік на сервері з можливістю відкликання (вихід з системи); застарілі або відкликані сесії не повинні прийматись для оновлення доступу.

У разі відмови зовнішніх компонентів (база даних, мережа) застосунок має коректно повідомляти про неможливість виконання операції;

час відновлення роботи визначається часом відновлення роботи інфраструктури та перезапуску серверних компонентів.

А.3.3 Умови експлуатації

Застосунок експлуатується у приміщеннях, придатних для роботи з ПК (температура навколишнього повітря для обладнання клієнтських робочих місць та сервера – згідно з паспортами техніки, зазвичай від +10 °С до +35 °С; відносна вологість до 80 % за відсутності конденсату).

Експлуатаційне супроводження передбачає планові оновлення програмних компонентів, регулярні резервні копії даних, контроль доступності серверної частини, а також керування життєвим циклом облікових записів у межах політики безпеки. Ці функції покладаються на адміністратора системи або уповноважену особу з відповідним рівнем доступу.

Кінцеві користувачі – працівники сервісного центру, яким для ефективної роботи достатньо навичок роботи з вебінтерфейсом та базових правил введення й перевірки даних. Адміністратор відповідає за облікові записи, конфігурацію середовища, підтримку стабільної роботи серверної частини й сховища даних, а також за типові процедури відновлення після збоїв і мінімізацію тривалих простоїв.

А.3.4 Вимоги до складу параметрів технічних засобів

Вимоги до робочого місця користувача:

- персональний комп'ютер або ноутбук із підключенням до мережі;
- операційна система: сучасна ОС сімейства Windows / Linux / macOS;
- веббраузер останніх стабільних версій (наприклад, на базі Chromium, Firefox, Safari);

- оперативна пам'ять: не менше 4 Гб для комфортної роботи в браузері з іншими застосунками;
- роздільна здатність екрана: не менше 1280×720 для коректного відображення інтерфейсу.

Вимоги до сервера та бази даних:

- процесор: x86-64, 2 ядра і більше;
- оперативна пам'ять: не менше 4–8 Гб залежно від обсягу даних і числа одночасних користувачів;
- дисковий простір: з урахуванням обсягу бази даних та журналів – не менше 20 Гб вільного місця для базової експлуатації;
- мережева взаємодія: доступ клієнтів до цього вузла в мережі.

A.3.5 Вимоги до інформаційної й програмної сумісності

Клієнт і сервер мають взаємодіяти через мережевий програмний інтерфейс із використанням узгодженого формату запитів і відповідей. Формат обміну має бути зручним для вебклієнта та придатним для перевірки на сервері.

Серверна частина має забезпечувати обробку бізнес-операцій, перевірку вхідних даних, взаємодію зі сховищем даних і повернення структурованих відповідей клієнту. Доступ до даних має виконуватись через узгоджений шар роботи зі сховищем, щоб зменшити залежність інтерфейсу від фізичної структури даних.

Користувацький інтерфейс має працювати у веббраузері, підтримувати маршрутизацію між основними екранами, відображення таблиць і форм, а також повідомлення про успіх або помилки. Візуальне оформлення має бути достатньо уніфікованим для типових дій користувача.

У робочому розгортанні обмін між браузером і сервером має виконуватись захищеним каналом. Службові секрети та параметри доступу

мають зберігатися лише в серверній конфігурації й не потрапляти до клієнтської частини.

А.3.6 Вимоги до маркування й упакування

Програмний продукт поширюється в електронному вигляді (вихідний код і збірки, передані замовнику). Вимоги до фізичного маркування, ярликів і упаковки носіїв не пред'являються. У супровідній документації зазначаються найменування «Сервісний центр», версія (за наявності) і відомості про розробника.

А.3.7 Вимоги до транспортування й збереження

Передбачається зберігання електронних носіїв із підсумковою документацією й резервними копіями бази даних у приміщеннях, захищених від несанкціонованого доступу, пилу й вологи, з дотриманням правил зберігання носіїв інформації. Вимоги до транспортування визначаються правилами обміну між розробником і замовником (передача через захищені канали або видані носії).

А.4 Вимоги до програмної документації

Програмна документація до застосунку включає технічне завдання (додаток до пояснювальної записки); пояснювальну записку до випускної кваліфікаційної роботи з описом аналізу предметної області, архітектури, реалізації й результатів тестування; текст програми (фрагменти та перелік ключових модулів); коментарі у коді на рівні, достатньому для розуміння основних алгоритмів і контрактів серверних інтерфейсів..

A.5 Техніко-економічні показники

Орієнтовна економічна ефективність впровадження полягає в скороченні часу на ручний облік замовлень і пошук інформації про клієнтів, історію ремонтів і використані матеріали, зниженні ризику помилок при оформленні заявок та підвищенні прозорості статусів замовлень для персоналу сервісу.

Річна потреба залежить від масштабу сервісного центру (число майстрів і щоденний потік замовлень); для навчального проєкту передбачається підтримка десятків тисяч записів у базі на стандартному апаратному забезпеченні, описаному в п. А.3.4.

Порівняно з веденням обліку лише в табличних редакторах або «на папері» розроблюваний застосунок забезпечує централізоване сховище, єдині правила доступу й можливість подальшого масштабування функцій без дублювання даних.

A.6 Стадії й етапи розробки

Розробка включає наступні етапи:

- постановка завдання (1 тиждень);
- аналіз предметної області (1–2 тижні);
- розробка архітектури (1 тиждень);
- розробка програми (1–2 тижні);
- тестування (1 тиждень);
- оформлення документації (1 тиждень).

A.7 Порядок контролю та приймання

Процес перевірки працездатності застосунку «Сервісний центр» передбачає послідовне проходження кількох етапів тестування. Модульне

тестування дає змогу переконатися, що окремі функції програми – зокрема автентифікація, валідація даних і операції з сутностями предметної області – працюють передбачувано й стабільно. Інтеграційне тестування зосереджене на взаємодії клієнта й сервера: коректності серверних інтерфейсів, оновлення сесії доступу та узгодженості відображення даних із тим, що повертає сервер. Окремо доцільно виконати перевірку вебінтерфейсу користувача, щоб переконатися у зрозумілості сценаріїв входу, роботи з формами та таблицями й адекватності повідомлень про помилки.

Працездатність вебзастосунку перевіряється у сучасних браузерах на типових робочих місцях користувача, а також у ситуаціях нестабільного мережевого з'єднання або недоступності сервера чи бази даних, коли важливо, щоб інтерфейс не залишав користувача без зрозумілих повідомлень про причину відмови й коректно супроводжував відновлення роботи після усунення збою. Після успішного завершення зазначених етапів перевірки та демонстрації відповідності вимогам технічного завдання програмний виріб разом із документацією передається замовнику для остаточного приймання.

ДОДАТОК Б
Текст програми

Б.1 Файл «server/db.js»

У файлі зосереджено налаштування з'єднання серверної частини з PostgreSQL через TypeORM, підключення сутностей проєкту та службовий механізм виконання запитів у межах визначеної ролі бази даних.

```
const path = require('path');
const { DataSource } = require('typeorm');
const { userEntity } = require('./entities/authEntity');
const { allBusinessEntitySchemas } = require('./entities/businessEntities');
require('dotenv').config({ path: path.resolve(__dirname, './.env') });

const appEntities = [userEntity, ...allBusinessEntitySchemas];

const AppDataSource = new DataSource({
  type: 'postgres',
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  username: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  synchronize: false,
  logging: true,
  entities: appEntities,
});

const initializeDataSources = async () => {
  try {
    if (!AppDataSource.isInitialized) {
      await AppDataSource.initialize();
      console.log('AppDataSource has been initialized!');
    }
  } catch (err) {
    console.error('Error during AppDataSource initialization', err);
  }
};

const queryWithRole = async (dbRole, queryFn) => {
  const queryRunner = AppDataSource.createQueryRunner();
  try {
    await queryRunner.connect();
    await queryRunner.query(`SET ROLE "${dbRole}"`);
    const result = await queryFn(queryRunner);
  }
```

```

    await queryRunner.query('RESET ROLE');
    return result;
  } finally {
    try {
      await queryRunner.query('RESET ROLE');
    } catch (e) { }
    await queryRunner.release();
  }
};

initializeDataSources();

module.exports = { AppDataSource, initializeDataSources, queryWithRole };

```

Б.2 Файл «server/index.js»

Файл є точкою входу серверної частини: у ньому налаштовано Express-додаток, маршрути автентифікації, перевірку JWT, refresh-сесії, рольові обмеження та підключення ресурсів предметної області.

```

const express = require('express');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const cors = require('cors');
const cookieParser = require('cookie-parser');
const crypto = require('crypto');
const cron = require('node-cron');
const path = require('path');
const { z } = require('zod');
const { AppDataSource, queryWithRole } = require('./db');
const { BaseService } = require('./services/baseService');
const { ReadOnlyBaseService } = require('./services/readOnlyBaseService');
const businessEntities = require('./entities/businessEntities');
const { userEntity, loginSchema, registerUserSchema, deleteUserParamsSchema, mapZodErrors } = require('./entities/authEntity');

// Завантаження змінних середовища з файлу .env
require('dotenv').config({ path: path.resolve(__dirname, './.env') });
const JWT_SECRET = process.env.JWT_SECRET;
const JWT_EXPIRES_IN = process.env.JWT_EXPIRES_IN || '30m';
const REFRESH_TOKEN_SECRET =
process.env.REFRESH_TOKEN_SECRET;

```

```

const REFRESH_TOKEN_EXPIRES_IN =
process.env.REFRESH_TOKEN_EXPIRES_IN || '7d';
const REFRESH_TOKEN_COOKIE_MAX_AGE_MS =
Number(process.env.REFRESH_TOKEN_COOKIE_MAX_AGE_MS || 7 * 24 *
60 * 60 * 1000);
const CORS_ORIGIN = process.env.CORS_ORIGIN || 'http://localhost:5173';
const IS_PROD = process.env.NODE_ENV === 'production';
const REFRESH_SESSIONS_CLEANUP_CRON =
process.env.REFRESH_SESSIONS_CLEANUP_CRON || '0 3 * * *';

// Перевірка обов'язкових змінних середовища
if (!JWT_SECRET) {
  console.error('JWT_SECRET is not defined');
  process.exit(1);
}
if (!REFRESH_TOKEN_SECRET) {
  console.error('REFRESH_TOKEN_SECRET is not defined');
  process.exit(1);
}

// Налаштування куки для refresh token
const REFRESH_COOKIE_NAME = 'refreshToken';
const REFRESH_COOKIE_OPTIONS = {
  httpOnly: true,
  secure: IS_PROD,
  sameSite: 'strict',
  path: '/api',
  maxAge: REFRESH_TOKEN_COOKIE_MAX_AGE_MS
};

// Створення Express-додатка
const app = express();
app.use(express.json());
app.use(cookieParser());
app.use(cors({
  origin: CORS_ORIGIN,
  credentials: true
}));

// Логування запитів
app.use((req, res, next) => {
  const timestamp = new Date().toISOString();
  const method = req.method;
  const path = req.path;
  console.log(`[${timestamp}] ${method} ${path}`);
});

```

```

    next();
  });

  // Надсилає успішну відповідь з опціональним повідомленням і даними
  const sendOk = (res, { message = "", data = null } = {}, statusCode = 200) => {
    return res.status(statusCode).json({ ok: true, message, data });
  };

  // Надсилає помилку з кодом, повідомленням і деталями
  const sendError = (res, { code, message, details = [] }, statusCode) => {
    return res.status(statusCode).json({ ok: false, code, message, details });
  };

  // Middleware для валідації запитів за допомогою схем з Zod
  const validateRequest = ({ body, params, query } = {}) => (req, res, next) => {
    const validations = [
      { target: 'body', schema: body, payload: req.body },
      { target: 'params', schema: params, payload: req.params },
      { target: 'query', schema: query, payload: req.query },
    ];

    for (const item of validations) {
      if (!item.schema) continue;
      const result = item.schema.safeParse(item.payload);
      if (!result.success) {
        return sendError(
          res,
          {
            code: 'VALIDATION_ERROR',
            message: 'Помилка валідації даних',
            details: mapZodErrors(result.error),
          },
          400
        );
      }
      req[item.target] = result.data;
    }

    return next();
  };

  // Middleware для перевірки JWT токена
  const verifyToken = (req, res, next) => {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];

```

```

    if (!token) {
      return sendError(res, { code: 'UNAUTHORIZED', message: 'Відсутній access token' }, 401);
    }
    jwt.verify(token, JWT_SECRET, (err, user) => {
      if (err) {
        return sendError(res, { code: 'FORBIDDEN', message: 'Невалідний або протермінований access token' }, 403);
      }
      req.user = user;
      next();
    });
  };

  // Middleware для перевірки ролей
  const requireRole = (...allowedRoles) => {
    return (req, res, next) => {
      if (!req.user) {
        return sendError(res, { code: 'UNAUTHORIZED', message: 'Не авторизовано' }, 401);
      }
      if (!allowedRoles.includes(req.user.role)) {
        return sendError(res, { code: 'FORBIDDEN', message: 'Недостатньо прав' }, 403);
      }
      next();
    };
  };

  // Створює новий access token для користувача
  const createAccessToken = (user) => {
    return jwt.sign({ userId: user.id, email: user.email, role: user.role }, JWT_SECRET, { expiresIn: JWT_EXPIRES_IN });
  };

  // Створює новий refresh token для користувача
  const createRefreshToken = (user) => {
    const jti = crypto.randomUUID();
    const refreshToken = jwt.sign(
      { userId: user.id, role: user.role, type: 'refresh', jti },
      REFRESH_TOKEN_SECRET,
      { expiresIn: REFRESH_TOKEN_EXPIRES_IN }
    );
    return { refreshToken, jti };
  };

```

```

// Хешування токєну
const hashToken = (token) =>
  crypto.createHash('sha256').update(token).digest('hex');

// Зберігає сесію refresh token у базі даних
const saveRefreshSession = async ({ userId, refreshToken, expiresAt, req }) => {
  const tokenHash = hashToken(refreshToken);
  const userAgent = req.get('user-agent') || null;
  const ipAddress = req.ip || req.socket?.remoteAddress || null;

  await AppDataSource.query(
    `INSERT INTO auth_refresh_sessions (user_id, token_hash, expires_at,
    user_agent, ip_address)
      VALUES ($1, $2, $3, $4, $5)`,
    [userId, tokenHash, expiresAt, userAgent, ipAddress]
  );
};

// Деактивує (відкликає) сесію refresh token
const revokeRefreshSession = async (refreshToken) => {
  const tokenHash = hashToken(refreshToken);
  await AppDataSource.query(
    `UPDATE auth_refresh_sessions
      SET revoked_at = NOW()
      WHERE token_hash = $1 AND revoked_at IS NULL`,
    [tokenHash]
  );
};

// Очищає старі та відкликані refresh-сесії з бази даних
const cleanupRefreshSessions = async () => {
  const result = await AppDataSource.query(
    `DELETE FROM auth_refresh_sessions
      WHERE expires_at < NOW()
      OR (revoked_at IS NOT NULL AND revoked_at < NOW() -
    INTERVAL '7 days')`
  );
  return result;
};

// Оновлення (refresh) токєну користувача
app.post('/api/refresh', async (req, res, next) => {
  const refreshToken = req.cookies[REFRESH_COOKIE_NAME];
  if (!refreshToken) {

```

```

    return sendError(res, { code: 'UNAUTHORIZED', message: 'Відсутній refresh
token' }, 401);
}

try {
    const payload = jwt.verify(refreshToken, REFRESH_TOKEN_SECRET);
    if (payload.type !== 'refresh') {
        return sendError(res, { code: 'FORBIDDEN', message: 'Невалідний тип
токена' }, 403);
    }

    const tokenHash = hashToken(refreshToken);
    const [session] = await AppDataSource.query(
        `SELECT id, user_id, expires_at, revoked_at
        FROM auth_refresh_sessions
        WHERE token_hash = $1
        LIMIT 1`,
        [tokenHash]
    );

    if (!session || session.revoked_at || new Date(session.expires_at) < new Date()) {
        return sendError(res, { code: 'FORBIDDEN', message: 'Refresh сесія
невалідна' }, 403);
    }

    const repository = AppDataSource.getRepository(userEntity);
    const user = await repository.findOne({
        where: { id: payload.userId },
        select: ['id', 'email', 'role']
    });

    if (!user) {
        return sendError(res, { code: 'NOT_FOUND', message: 'Користувач не
знайдений' }, 404);
    }

    await revokeRefreshSession(refreshToken);
    const newAccessToken = createAccessToken(user);
    const { refreshToken: newRefreshToken } = createRefreshToken(user);
    const decodedRefresh = jwt.decode(newRefreshToken);
    const expiresAt = new Date(decodedRefresh.exp * 1000);
    await saveRefreshSession({ userId: user.id, refreshToken: newRefreshToken,
expiresAt, req });

```

```

    res.cookie(REFRESH_COOKIE_NAME, newRefreshToken,
REFRESH_COOKIE_OPTIONS);
    return sendOk(res, { message: 'Сесію оновлено', data: { token:
newAccessToken, user } });
  } catch (err) {
    if (err instanceof jwt.JsonWebTokenError || err instanceof
jwt.TokenExpiredError) {
      return sendError(res, { code: 'FORBIDDEN', message: 'Невалідний або
протермінований refresh token' }, 403);
    }
    return next(err);
  }
});

// Відкликає refresh-сесію та очищає cookie
app.post('/api/logout', async (req, res) => {
  const refreshToken = req.cookies[REFRESH_COOKIE_NAME];
  if (refreshToken) {
    await revokeRefreshSession(refreshToken);
  }
  res.clearCookie(REFRESH_COOKIE_NAME,
REFRESH_COOKIE_OPTIONS);
  return sendOk(res, { message: 'Вихід виконано' });
});

// Отримати інформацію про поточного користувача
app.get('/api/me', verifyToken, async (req, res) => {
  return sendOk(res, {
    message: 'Дані поточного користувача',
    data: {
      id: req.user.userId,
      email: req.user.email,
      role: req.user.role
    }
  });
});

// CRON-задача для очищення протермінованих refresh-сесій
cron.schedule(REFRESH_SESSIONS_CLEANUP_CRON, async () => {
  try {
    const result = await cleanupRefreshSessions();
    const deletedRows = Array.isArray(result) ? result[1] ?? 'unknown' : 'unknown';
    console.log(`[CRON] Refresh sessions cleanup completed. Deleted rows:
${deletedRows}`);
  } catch (error) {

```

```

    console.error('[CRON] Refresh sessions cleanup failed:', error);
  }
});

// Реєстрація (тільки адмін)
app.post('/api/register', verifyToken, requireRole('admin'), validateRequest({ body:
registerUserSchema })), async (req, res, next) => {
  const { email, password } = req.body;
  try {
    const hashedPassword = await bcrypt.hash(password, 10);
    const result = await queryWithRole(req.user.role, async (queryRunner) => {
      const repository = queryRunner.manager.getRepository(userEntity);
      const newUser = repository.create({
        email,
        password_hash: hashedPassword,
        role: 'app_user'
      });
      const savedUser = await repository.save(newUser);
      return {
        id: savedUser.id,
        email: savedUser.email,
        role: savedUser.role
      };
    });
    return sendOk(res, { message: 'Користувача створено', data: result }, 201);
  } catch (err) {
    if (err.code === '23505') {
      return sendError(
        res,
        { code: 'CONFLICT', message: 'Користувач з таким email вже існує' },
        409
      );
    }
  }
  return next(err);
}
});

// Вхід
app.post('/api/login', validateRequest({ body: loginSchema })), async (req, res,
next) => {
  const { email, password } = req.body;
  try {
    const repository = AppDataSource.getRepository(userEntity);
    const user = await repository.findOne({ where: { email } });

```

```

    if (user && (await bcrypt.compare(password, user.password_hash))) {
      const token = createAccessToken(user);
      const { refreshToken } = createRefreshToken(user);
      const decodedRefresh = jwt.decode(refreshToken);
      const expiresAt = new Date(decodedRefresh.exp * 1000);

      await saveRefreshSession({ userId: user.id, refreshToken, expiresAt, req });
      res.cookie(REFRESH_COOKIE_NAME, refreshToken,
REFRESH_COOKIE_OPTIONS);
      return sendOk(res, {
        message: 'Успішний вхід',
        data: { token, user: { id: user.id, email: user.email, role: user.role } }
      });
    } else {
      return sendError(
        res,
        { code: 'UNAUTHORIZED', message: 'Невірний email або пароль' },
        401
      );
    }
  } catch (err) {
    return next(err);
  }
});

// Всі користувачі (тільки адмін)
app.get('/api/users', verifyToken, requireRole('admin'), async (req, res, next) => {
  try {
    const result = await queryWithRole(req.user.role, async (queryRunner) => {
      const repository = queryRunner.manager.getRepository(userEntity);
      const users = await repository.find({
        select: ['id', 'email', 'role']
      });
      return users;
    });

    sendOk(res, {
      message: 'Користувачі отримані',
      data: result
    });
  } catch (err) {
    console.error('Помилка при отриманні користувачів:', err);
    return next(err);
  }
});

```

```

// Видалити користувача (тільки адмін)
app.delete('/api/users/:id', verifyToken, requireRole('admin'), validateRequest({
  params: deleteUserParamsSchema }), async (req, res, next) => {
  try {
    const { id } = req.params;

    // Забезпечуємо, щоб адмін не видалив себе
    if (id === req.user.userId) {
      return sendError(res, { code: 'CONFLICT', message: 'Ви не можете видалити
свій акаунт' }, 409);
    }

    const result = await queryWithRole(req.user.role, async (queryRunner) => {
      const repository = queryRunner.manager.getRepository(userEntity);
      const user = await repository.findOne({ where: { id } });

      if (!user) {
        return { found: false };
      }

      await repository.delete({ id });
      return { found: true, user };
    });

    if (!result.found) {
      return sendError(res, { code: 'NOT_FOUND', message: 'Користувач не
знайдений' }, 404);
    }

    sendOk(res, {
      message: 'Користувач видалений',
      data: result.user
    });
  } catch (err) {
    console.error('Помилка при видаленні користувача:', err);
    return next(err);
  }
});

// Zod-схема для валідації id-параметра
const defaultCrudIdParamSchema = z.object({
  id: z.coerce.number().int().positive('ID має бути додатним числом'),
});

```

```
// З одного ресурсного Zod-об'єкта буде тіло створення та оновлення (без
генерованих id / id_*).
function resourceZodToCreatePatch(resourceSchema) {
  const shape = resourceSchema.shape || {};
  const omit = {};
  if (shape.id) omit.id = true;
  for (const key of Object.keys(shape)) {
    if (key.startsWith('id_')) omit[key] = true;
  }
  const createBody =
    Object.keys(omit).length > 0 ? resourceSchema.omit(omit) : resourceSchema;
  return { createBody, patchBody: createBody.partial() };
}
```

// Нормалізує другий аргумент: або об'єкт з полями
create/patch/idParam/listQuery, або один Zod object з «повною» схемою
ресурсу.

```
function resolveCrudZod(zodSchema) {
  if (zodSchema && typeof zodSchema === 'object' && typeof
zodSchema.safeParse !== 'function') {
    const c = zodSchema.create ?? zodSchema.createBody;
    const p = zodSchema.patch ?? zodSchema.patchBody;
    return {
      createBody: c,
      patchBody: p,
      idParam: zodSchema.idParam ?? defaultCrudIdParamSchema,
      listQuery: zodSchema.listQuery,
    };
  }
  const { createBody, patchBody } = resourceZodToCreatePatch(zodSchema);
  return {
    createBody,
    patchBody,
    idParam: defaultCrudIdParamSchema,
    listQuery: undefined,
  };
}
```

// Створює CRUD роути

```
function createCrudRouter(service, zodSchema) {
  const entity = service.repository.target;
  const { createBody, patchBody, idParam, listQuery } =
resolveCrudZod(zodSchema);
  const readOnly = service instanceof ReadOnlyBaseService;
```

```

const router = express.Router();

const scopedServiceOpts = { primaryKey: service.primaryKey || 'id' };

const runWithRole = (req, handler) =>
  queryWithRole(req.user.role, async (queryRunner) => {
    const repository = queryRunner.manager.getRepository(entity);
    const scoped = readOnly
      ? new ReadOnlyBaseService(repository, scopedServiceOpts)
      : new BaseService(repository, scopedServiceOpts);
    return handler(scoped);
  });

const wrapAsync = (fn) => (req, res, next) => Promise.resolve(fn(req, res,
next)).catch(next);

const listMiddleware = listQuery ? [validateRequest({ query: listQuery })]: [];

router.get(
  '/',
  ...listMiddleware,
  wrapAsync(async (req, res, next) => {
    try {
      const data = await runWithRole(req, (s) => s.findAll());
      sendOk(res, { message: 'Список отримано', data });
    } catch (err) {
      next(err);
    }
  })
);

router.get(
 ('/:id',
  validateRequest({ params: idParam }),
  wrapAsync(async (req, res, next) => {
    try {
      const row = await runWithRole(req, (s) => s.findById(req.params.id));
      if (!row) {
        return sendError(res, { code: 'NOT_FOUND', message: 'Запис не
знайдений' }, 404);
      }
      sendOk(res, { message: 'Запис отримано', data: row });
    } catch (err) {
      next(err);
    }
  })
);

```

```

    })
  );

  if (readOnly) {
    return router;
  }

  const postMiddlewares = createBody ? [validateRequest({ body: createBody })] :
  [];

  router.post(
    '/',
    ...postMiddlewares,
    wrapAsync(async (req, res, next) => {
      try {
        const row = await runWithRole(req, (s) => s.create(req.body));
        sendOk(res, { message: 'Запис створено', data: row }, 201);
      } catch (err) {
        if (err.code === '23505') {
          return sendError(
            res,
            {
              code: 'CONFLICT',
              message:
                err.detail?.slice(0, 200) ||
                'Запис з такими значеннями вже існує',
            },
            409
          );
        }
        next(err);
      }
    })
  );

  const patchMiddlewares =
    patchBody != null
      ? [validateRequest({ params: idParam, body: patchBody })]
      : [];
  router.patch(
    `/:id`,
    ...patchMiddlewares,
    wrapAsync(async (req, res, next) => {
      if (patchBody == null) {
        return sendError(

```

```

    res,
    { code: 'METHOD_NOT_ALLOWED', message: 'Оновлення не
дозволено' },
    405
  );
}
try {
  const row = await runWithRole(req, (s) =>
    s.update(req.params.id, req.body)
  );
  if (!row) {
    return sendError(res, { code: 'NOT_FOUND', message: 'Запис не
знайдений' }, 404);
  }
  sendOk(res, { message: 'Запис оновлено', data: row });
} catch (err) {
  if (err.code === '23505') {
    return sendError(
      res,
      {
        code: 'CONFLICT',
        message:
          err.detail?.slice(0, 200) ||
          'Запис з такими значеннями вже існує',
      },
      409
    );
  }
  next(err);
}
})
);

router.delete(
 ('/:id',
  validateRequest({ params: idParam }),
  wrapAsync(async (req, res, next) => {
    try {
      const ok = await runWithRole(req, (s) => s.remove(req.params.id));
      if (!ok) {
        return sendError(res, { code: 'NOT_FOUND', message: 'Запис не
знайдений' }, 404);
      }
      sendOk(res, { message: 'Запис видалено', data: null });
    } catch (err) {

```

```

        next(err);
    }
})
);

return router;
}

// Захист доступу до бізнес-ресурсів
const requireBusinessAccess = requireRole('admin', 'app_user');

// Мінімальна схема для read-only view (лише GET; POST/PATCH не
реєструються)
const viewReadOnlyStubZod = z.object({
    id: z.number().int().positive().optional(),
});

/** Параметр `/:id` для view з ключем-датою `month_start_date` (YYYY-MM-DD). */
const monthlyOrdersStatisticsViewIdParam = z.object({
    id: z
        .string()
        .regex(/^\d{4}-\d{2}-\d{2}$/, 'Має бути дата у форматі YYYY-MM-DD'),
});

const businessTableCrud = [
    ['/api/clients', businessEntities.clientsEntity, businessEntities.clientsZodSchema],
    ['/api/employees', businessEntities.employeesEntity,
businessEntities.employeesZodSchema],
    ['/api/orders', businessEntities.ordersEntity, businessEntities.ordersCrudZod],
    ['/api/services', businessEntities.servicesEntity,
businessEntities.servicesZodSchema],
    ['/api/spare-parts', businessEntities.sparePartsEntity,
businessEntities.sparePartsZodSchema],
    ['/api/ordered-services', businessEntities.orderedServicesEntity,
businessEntities.orderedServicesCrudZod],
    ['/api/ordered-spare-parts', businessEntities.orderedSparePartsEntity,
businessEntities.orderedSparePartsCrudZod],
];

for (const [routePath, entity, schema] of businessTableCrud) {
    app.use(
        routePath,
        verifyToken,
        requireBusinessAccess,

```

```

    createCrudRouter(new BaseService(AppDataSource.getRepository(entity)),
schema)
  );
}

const businessViews = [
  ['/api/clients-view', businessEntities.clientsViewEntity],
  ['/api/orders-view', businessEntities.ordersViewEntity],
  ['/api/employers-view', businessEntities.employersViewEntity],
  ['/api/ordered-services-view', businessEntities.orderedServicesViewEntity],
  ['/api/ordered-spare-parts-view', businessEntities.orderedSparePartsViewEntity],
];

for (const [routePath, entity] of businessViews) {
  app.use(
    routePath,
    verifyToken,
    requireBusinessAccess,
    createCrudRouter(
      new ReadOnlyBaseService(AppDataSource.getRepository(entity)),
      viewReadOnlyStubZod
    )
  );
}

app.use(
  '/api/monthly-orders-statistics',
  verifyToken,
  requireBusinessAccess,
  createCrudRouter(
    new ReadOnlyBaseService(
AppDataSource.getRepository(businessEntities.monthlyOrdersStatisticsViewEntit
y),
    { primaryKey: 'month_start_date' }
  ),
  { idParam: monthlyOrdersStatisticsViewIdParam }
)
);

app.use((err, req, res, next) => {
  console.error('Неперехоплена помилка:', err);
  return sendError(res, { code: 'INTERNAL_ERROR', message: 'Помилка
сервера' }, 500);
});

```

```
const PORT = 5000;
app.listen(PORT, () => console.log(`Сервер запущено на порту ${PORT}`));
```

Б.3 Файл «server/services/baseService.js»

Файл описує спільний сервісний шар для типових CRUD-операцій, завдяки чому однотипна робота з таблицями бази даних не дублюється в різних маршрутах API.

```
/**
 * Базові операції з репозиторієм
 * @template T
 */
class BaseService {
  constructor(repository, options = {}) {
    this.repository = repository;
    this.primaryKey = options.primaryKey || 'id';
  }

  findAll(options) {
    return this.repository.find(options);
  }

  findById(id, options = {}) {
    return this.repository.findOne({
      where: { [this.primaryKey]: id },
      ...options,
    });
  }

  create(data) {
    const entity = this.repository.create(data);
    return this.repository.save(entity);
  }

  async update(id, partial) {
    const row = await this.findById(id);
    if (!row) return null;
    Object.assign(row, partial);
    return this.repository.save(row);
  }
}
```

```

    async remove(id) {
      const result = await this.repository.delete({ [this.primaryKey]: id });
      return (result.affected ?? 0) > 0;
    }
  }

  module.exports = { BaseService };

```

Б.4 Файл «server/services/readOnlyBaseService.js»

У файлі подано сервіс для читання даних із подань PostgreSQL, які використовуються в інтерфейсі для зведеного перегляду без виконання операцій зміни через ці подання.

```

/**
 * Лише читання (view / репозиторії без мутацій).
 * @template T
 */
class ReadOnlyBaseService {
  constructor(repository, options = {}) {
    this.repository = repository;
    this.primaryKey = options.primaryKey || 'id';
  }

  findAll(options) {
    return this.repository.find(options);
  }

  findById(id, options = {}) {
    return this.repository.findOne({
      where: { [this.primaryKey]: id },
      ...options,
    });
  }
}

module.exports = { ReadOnlyBaseService };

```

Б.5 Файл «server/entities/authEntity.js»

Файл визначає модель користувача застосунку, перелік ролей доступу та схеми перевірки даних для входу, створення й видалення облікових записів.

```
const { EntitySchema } = require('typeorm');  
const { z } = require('zod');
```

```
const USER_ROLES = ['admin', 'app_user'];
```

```
const userEntity = new EntitySchema({  
  name: 'User',  
  tableName: 'users',  
  columns: {  
    id: {  
      primary: true,  
      type: 'int',  
      generated: true,  
    },  
    email: {  
      type: 'varchar',  
      length: 255,  
      nullable: false,  
      unique: true,  
    },  
    password_hash: {  
      type: 'varchar',  
      length: 255,  
      nullable: false,  
    },  
    role: {  
      type: 'enum',  
      enum: USER_ROLES,  
      enumName: 'user_role',  
      default: 'app_user',  
      nullable: false,  
    },  
  },  
});
```

```
const userSchema = z.object({  
  id: z.number().int().optional(),
```

```

email: z.string().email('Некоректний формат email').max(255),
password_hash: z.string().min(6, 'Пароль занадто короткий').max(255),
role: z.enum(USER_ROLES).default('app_user'),
});

const loginSchema = z.object({
  email: z.string().email('Некоректний формат email').max(255),
  password: z.string().min(6, 'Пароль занадто короткий').max(255),
});

const registerUserSchema = z.object({
  email: z.string().email('Некоректний формат email').max(255),
  password: z.string().min(6, 'Пароль занадто короткий').max(255),
});

const deleteUserParamsSchema = z.object({
  id: z.coerce.number().int().positive('ID користувача має бути додатним числом'),
});

const mapZodErrors = (error) => {
  const rawIssues = error?.issues || error?.errors || [];
  return rawIssues.map((e) => ({ field: e.path.join('.'), message: e.message }));
};

module.exports = {
  userEntity,
  USER_ROLES,
  loginSchema,
  registerUserSchema,
  deleteUserParamsSchema,
  mapZodErrors,
};

```

Б.6 Файл «server/entities/businessEntities/orders.js»

У файлі описано ключову сутність предметної області – замовлення, включно з її полями, статусом виконання та правилами перевірки даних перед записом у базу.

```

const { EntitySchema } = require('typeorm');
const { z } = require('zod');

```

```
const { STATUS_ORDER } = require('./statusOrder');
```

```
const ordersEntity = new EntitySchema({
  name: 'Order',
  tableName: 'orders',
  columns: {
    id: { type: 'int', primary: true, generated: true },
    id_order: { type: 'int', unique: true, generated: true },
    id_client: { type: 'int', nullable: false },
    name_device: { type: 'varchar', length: 255, nullable: false },
    description: { type: 'varchar', length: 1000, nullable: true },
    id_emp: { type: 'int', nullable: true },
    date_creation: { type: 'timestamp without time zone', nullable: false },
    date_end: { type: 'timestamp without time zone', nullable: true },
    price_order: { type: 'int', default: 0, nullable: false },
    status_order: {
      type: 'enum',
      enum: STATUS_ORDER,
      enumName: 'status_order',
      default: 'Прийнято',
      nullable: false,
    },
  },
});
```

/** Об'єкт без крос-схемного refine – для `.omit()` у createCrudRouter (Zod v4 забороняє omit після `.refine()`). */

```
const ordersShape = z.object({
  id: z.number().int().positive().optional(),
  id_order: z.number().int().positive().optional(),
  id_client: z.number().int().positive(),
  name_device: z.string().min(1).max(255),
  description: z.string().max(1000).nullish(),
  id_emp: z.number().int().positive().nullish(),
  date_creation: z.coerce.date().optional(),
  date_end: z.coerce.date().nullish(),
  price_order: z.number().int().min(0),
  status_order: z.enum(STATUS_ORDER).default('Прийнято'),
});
```

```
const ordersZodSchema = ordersShape.refine(
  (row) =>
    row.date_end == null ||
    row.date_creation == null ||
    row.date_end >= row.date_creation,
```

```

    { message: 'date_end має бути >= date_creation', path: ['date_end'] }
  );

  /** Лише `id` та згенерований `id_order` вилучаються; `id_client` / `id_emp`
  лишаються (на відміну від загального `id_*` omit у CRUD). */
  const ordersDateRefine = (row) =>
    row.date_end == null ||
    row.date_creation == null ||
    row.date_end >= row.date_creation;

  const ordersCrudZod = {
    createBody: ordersShape
      .omit({ id: true, id_order: true })
      .refine(ordersDateRefine, {
        message: 'date_end має бути >= date_creation',
        path: ['date_end'],
      }),
    patchBody: ordersShape
      .omit({ id: true, id_order: true })
      .partial()
      .refine(ordersDateRefine, {
        message: 'date_end має бути >= date_creation',
        path: ['date_end'],
      }),
  };

  module.exports = { ordersEntity, ordersShape, ordersZodSchema, ordersCrudZod };

```

Б.7 Файл «client/src/App.jsx»

Файл задає загальну структуру клієнтської частини: маршрути сторінок, підключення контексту автентифікації та розмежування відкритих і захищених екранів.

```

import React from 'react';
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom';
import 'bootstrap/dist/css/bootstrap.min.css';
import './styles/appPageFlash.css';

import { AuthProvider } from './contexts/AuthContext';
import { ProtectedRoute } from './components/ProtectedRoute';

```

```

import { RoleRoute } from './components/RoleRoute';

import MainLayout from './layouts/MainLayout';
import LoginPage from './Pages/LoginPage';
import ForbiddenPage from './Pages/ForbiddenPage';
import UsersManagementPage from './Pages/UsersManagementPage';
import OrdersPage from './Pages/OrdersPage';
import OrderDetailPage from './Pages/OrderDetailPage';
import ClientsPage from './Pages/ClientsPage';
import EmployersPage from './Pages/EmployersPage';
import ServicesPage from './Pages/ServicesPage';
import SparePartsPage from './Pages/SparePartsPage';
import StatisticsPage from './Pages/StatisticsPage';

export default function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <Routes>
          { /* Публічні сторінки */ }
          <Route path="/login" element={ <LoginPage /> } />
          <Route path="/forbidden" element={ <ForbiddenPage /> } />

          { /* Захищені сторінки */ }
          <Route
            path="/"
            element={
              <ProtectedRoute>
                <MainLayout />
              </ProtectedRoute>
            }
          />
          <Route index element={ <OrdersPage /> } />
          <Route path="orders/:idOrder" element={ <OrderDetailPage /> } />
          <Route path="orders" element={ <OrdersPage /> } />
          <Route path="clients" element={ <ClientsPage /> } />
          <Route path="employees" element={ <EmployersPage /> } />
          <Route path="services" element={ <ServicesPage /> } />
          <Route path="spare-parts" element={ <SparePartsPage /> } />
          <Route path="statistics" element={ <StatisticsPage /> } />
          <Route
            path="users"
            element={
              <RoleRoute requiredRoles="admin">
                <UsersManagementPage />
              </RoleRoute>
            }
          />
        </Routes>
      </BrowserRouter>
    </AuthProvider>
  );
}

```

```

        </RoleRoute>
      }
    />
  </Route>

  { /* Незнайомий маршрут */ }
  <Route path="*" element={ <Navigate to="/" replace /> } />
</Routes>
</BrowserRouter>
</AuthProvider>
);
}

```

Б.8 Файл «client/src/contexts/AuthContext.jsx»

У файлі реалізовано клієнтський контекст автентифікації, який зберігає стан поточного користувача, відновлює сесію при відкритті застосунку та забезпечує вхід і вихід із системи.

```

import React, { createContext, useState, useContext, useEffect } from 'react';
import { authAPI } from '../api';
import { clearAccessToken, setAccessToken } from '../api/tokenStore';

```

```

const AuthContext = createContext();

```

```

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

```

```

  // Перевірка сесії при завантаженні
  useEffect(() => {
    const savedUser = localStorage.getItem('user');
    if (savedUser) {
      setUser(JSON.parse(savedUser));
    }

```

```

const bootstrapSession = async () => {
  try {
    const response = await authAPI.refresh();
    const { token, user: currentUser } = response.data.data;
    setAccessToken(token);
    setUser(currentUser);

```

```

    localStorage.setItem('user', JSON.stringify(currentUser));
  } catch {
    clearAccessToken();
    localStorage.removeItem('user');
    setUser(null);
  } finally {
    setLoading(false);
  }
};

bootstrapSession();
}, []);

const login = (userData, token) => {
  setAccessToken(token);
  localStorage.setItem('user', JSON.stringify(userData));
  setUser(userData);
};

const logout = async () => {
  try {
    await authAPI.logout();
  } catch {
    // Якщо сервер недоступний, локально все одно очищаємо сесію
  }
  clearAccessToken();
  localStorage.removeItem('user');
  setUser(null);
};

const value = {
  user,
  loading,
  login,
  logout,
  isAuthenticated: !!user,
};

return (
  <AuthContext.Provider value={value}>
    {children}
  </AuthContext.Provider>
);
};

```

```
export const useAuth = () => {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error('useAuth must be used within AuthProvider');
  }
  return context;
};
```

Б.9 Файл «client/src/Components/ProtectedRoute.jsx»

Файл містить компонент, що захищає основні сторінки застосунку від перегляду без успішної автентифікації користувача.

```
import React from 'react';
import { Navigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

export const ProtectedRoute = ({ children }) => {
  const { isAuthenticated, loading } = useAuth();
  if (loading) {
    return (
      <div className="d-flex align-items-center justify-content-center" style={{
minHeight: '100vh' }}>
        <div className="spinner-border text-primary" role="status">
          <span className="visually-hidden">Завантаження...</span>
        </div>
      </div>
    );
  }
  if (!isAuthenticated) {
    return <Navigate to="/login" replace />;
  }
  return children;
};
```

Б.10 Файл «client/src/Components/RoleRoute.jsx»

У файлі реалізовано перевірку ролі поточного користувача, яка не допускає відкриття адміністративних сторінок без відповідних прав доступу.

```
import React from 'react';
```

```

import { Navigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

export const RoleRoute = ({ children, requiredRoles }) => {
  const { user, loading, isAuthenticated } = useAuth();

  if (loading) {
    return (
      <div className="d-flex align-items-center justify-content-center" style={{
minHeight: '100vh' }}>
        <div className="spinner-border text-primary" role="status">
          <span className="visually-hidden">Завантаження...</span>
        </div>
      </div>
    );
  }

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

  // Якщо requiredRoles не передано, дозволити доступ
  if (!requiredRoles) {
    return children;
  }

  // Перевірити чи роль користувача входить в масив допустимих ролей
  const rolesArray = Array.isArray(requiredRoles) ? requiredRoles :
[requiredRoles];

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

  return children;
};

```

Б.11 Файл «client/src/api/client.js»

Файл налаштовує спільний axios-клієнт для HTTP-запитів, додає access-токен до захищених звернень і підтримує повтор запиту після успішного оновлення сесії.

```

import axios from 'axios';
import { clearAccessToken, getAccessToken, setAccessToken } from
'./tokenStore';

const api = axios.create({
  baseURL: 'http://localhost:5000',
  withCredentials: true,
});

let isRefreshing = false;
let refreshPromise = null;

api.interceptors.request.use((config) => {
  const token = getAccessToken();
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

api.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config;
    const status = error.response?.status;
    const url = originalRequest?.url || '';
    const isRefreshRequest = originalRequest?.url?.includes('/api/refresh');
    const isLoginOrRegisterRequest = url.includes('/api/login') ||
url.includes('/api/register');

    if (status !== 401 || originalRequest?._retry) {
      return handleUnauthorized(error);
    }
    if (isRefreshRequest || isLoginOrRegisterRequest) {
      // Для login/register повертаємо оригінальну помилку, не підміняємо її
refresh-помилкою
      return handleUnauthorized(error);
    }

    if (!isRefreshing) {
      isRefreshing = true;
      refreshPromise = api.post('/api/refresh')
        .then((response) => {
          const token = response?.data?.data?.token;
          if (!token) {

```

```

        throw new Error('Missing access token in refresh response');
    }
    setAccessToken(token);
    return token;
  })
  .finally(() => {
    isRefreshing = false;
  });
}

try {
  const newToken = await refreshPromise;
  originalRequest._retry = true;
  originalRequest.headers.Authorization = `Bearer ${newToken}`;
  return api(originalRequest);
} catch (refreshError) {
  return handleUnauthorized(refreshError);
}
);

const handleUnauthorized = (error) => {
  if (error.response?.status === 401 || error.response?.status === 403) {
    clearAccessToken();
    localStorage.removeItem('user');
    // Уникаємо циклічного reload на сторінці логіну
    if (window.location.pathname !== '/login') {
      window.location.href = '/login';
    }
  }
  return Promise.reject(error);
};

export default api;

```

Б.12 Файл «client/src/api/orders.js»

У файлі зібрано клієнтські звернення до API замовлень: отримання списку, читання окремого запису, створення, оновлення та видалення заявки через сервер.

```
import api from './client';
```

```
export const ordersAPI = {
  getById: (id) => api.get(`/api/orders/${id}`),
  create: (body) => api.post('/api/orders', body),
  patch: (id, body) => api.patch(`/api/orders/${id}`, body),
  remove: (id) => api.delete(`/api/orders/${id}`),
};
```

Б.13 Файл «client/src/Pages/OrdersPage.jsx»

Файл реалізує сторінку списку замовлень – один із головних робочих екранів оператора, з якого користувач переглядає заявки, виконує пошук і переходить до картки конкретного замовлення.

```
import React, { useCallback, useEffect, useMemo, useState } from 'react';
import {
  Container,
  Row,
  Col,
  Table,
  Spinner,
  Alert,
  Form,
  InputGroup,
  Modal,
  Button,
} from 'react-bootstrap';
import { ChevronUp, ChevronDown, PencilSquare, Trash } from 'react-bootstrap-icons';
import { useNavigate } from 'react-router-dom';
import { ordersViewAPI, ordersAPI, clientsViewAPI, employersViewAPI } from '../api';
import { parseApiResponse } from '../utils/parseApiResponse';
import OrderFormFields, {
  ORDER_STATUS_OPTIONS,
  emptyNewOrderForm,
} from '../components/order/OrderFormFields';

const SEARCH_CRITERIA = [
  { value: 'all', label: 'Усі поля' },
  { value: 'id_order', label: 'ID замовлення' },
  { value: 'id_client', label: 'ID клієнта' },
```

```

    { value: 'name_client', label: 'Клієнт' },
    { value: 'phone_client', label: 'Телефон' },
    { value: 'name_device', label: 'Пристрій' },
    { value: 'description', label: 'Опис' },
    { value: 'id_emp', label: 'ID майстра' },
    { value: 'name_emp', label: 'Майстер' },
    { value: 'status_order', label: 'Статус' },
  ];

```

```

function rowMatchesSearch(row, rowQuery, criterion) {
  const q = rowQuery.trim();
  if (!q) return true;

```

```

  if (criterion === 'all') {
    const bundle = [
      row.id_order,
      row.id_client,
      row.name_client,
      row.phone_client,
      row.name_device,
      row.description,
      row.id_emp,
      row.name_emp,
      row.status_order,
    ]
      .map((v) => String(v ?? "").toLowerCase())
      .join(' ');
    return bundle.includes(q.toLowerCase());
  }

```

```

  const val = row[criterion];
  if (val == null) return false;
  if (criterion === 'id_order' || criterion === 'id_client' || criterion === 'id_emp') {
    return String(val).includes(q);
  }
  return String(val).toLowerCase().includes(q.toLowerCase());
}

```

```

function formatDate(value) {
  if (value == null || value === "") return '-';
  const d = new Date(value);
  return Number.isNaN(d.getTime()) ? String(value) : d.toLocaleString('uk-UA');
}

```

```

function formatMoney(value) {

```

```

    if (value == null || value === '') return '-';
    return `${Number(value).toLocaleString('uk-UA')} ₴`;
}

const DATE_SORT_KEYS = new Set(['date_creation', 'estimated_date_end',
'date_end']);

const NUM_SORT_KEYS = new Set([
    'id_order',
    'id_client',
    'id_emp',
    'estimated_price',
    'price_order',
]);

/** Порівняння для сортування: null/undefined в кінці. */
function compareSortValues(a, b, key) {
    const va = a[key];
    const vb = b[key];
    if (va == null && vb == null) return 0;
    if (va == null) return 1;
    if (vb == null) return -1;

    if (DATE_SORT_KEYS.has(key)) {
        const ta = new Date(va).getTime();
        const tb = new Date(vb).getTime();
        const na = Number.isNaN(ta) ? 0 : ta;
        const nb = Number.isNaN(tb) ? 0 : tb;
        return na - nb;
    }

    if (NUM_SORT_KEYS.has(key)) {
        return Number(va) - Number(vb);
    }

    return String(va).localeCompare(String(vb), 'uk', { sensitivity: 'base' });
}

const TABLE_COLUMNS = [
    { sortKey: 'id_order', label: 'ID\нзам.' },
    { sortKey: 'id_client', label: 'ID\нкл.' },
    { sortKey: 'name_client', label: 'Клієнт' },
    { sortKey: 'phone_client', label: 'Телефон' },
    { sortKey: 'name_device', label: 'Пристрій' },
    { sortKey: 'description', label: 'Опис' },

```

```

    { sortKey: 'id_emp', label: 'ID\нм.' },
    { sortKey: 'name_emp', label: 'Майстер' },
    { sortKey: 'date_creation', label: 'Створено' },
    { sortKey: 'estimated_date_end', label: 'Орієнт.\нзавершення' },
    { sortKey: 'estimated_price', label: 'Орієнт.\нвартість' },
    { sortKey: 'date_end', label: 'Завершено' },
    { sortKey: 'price_order', label: 'Вартість' },
    { sortKey: 'status_order', label: 'Статус' },
  ];

export default function OrdersPage() {
  const navigate = useNavigate();
  const [allOrders, setAllOrders] = useState([]);
  const [listLoading, setListLoading] = useState(true);
  const [listError, setListError] = useState("");
  const [searchQuery, setSearchQuery] = useState("");
  const [searchCriterion, setSearchCriterion] = useState('all');
  const [sortKey, setSortKey] = useState(null);
  const [sortDir, setSortDir] = useState('asc');
  const [newOrder, setNewOrder] = useState(emptyNewOrderForm);
  const [formError, setFormError] = useState("");
  const [formSuccess, setFormSuccess] = useState("");
  const [creating, setCreating] = useState(false);
  const [clientsRows, setClientsRows] = useState([]);
  const [employersRows, setEmployersRows] = useState([]);
  const [lookupsLoading, setLookupsLoading] = useState(true);
  const [clientsLookupError, setClientsLookupError] = useState("");
  const [employersLookupError, setEmployersLookupError] = useState("");
  const [showNewOrderModal, setShowNewOrderModal] = useState(false);
  const [showEditOrderModal, setShowEditOrderModal] = useState(false);
  const [editingOrderId, setEditingOrderId] = useState(null);
  const [editingIdOrder, setEditingIdOrder] = useState(null);
  const [editOrder, setEditOrder] = useState(emptyNewOrderForm);
  const [editFormError, setEditFormError] = useState("");
  const [updating, setUpdating] = useState(false);
  const [deleteTarget, setDeleteTarget] = useState(null);
  const [deleting, setDeleting] = useState(false);
  const [deleteError, setDeleteError] = useState("");

  const closeNewOrderModal = useCallback(() => {
    setShowNewOrderModal(false);
    setFormError("");
  }, []);

  const closeEditOrderModal = useCallback(() => {

```

```

    setShowEditOrderModal(false);
    setEditFormError("");
    setEditingOrderId(null);
    setEditingIdOrder(null);
  }, []);

const closeDeleteModal = useCallback(() => {
  setDeleteTarget(null);
  setDeleteError("");
}, []);

const displayOrders = useMemo(() => {
  const filtered = allOrders.filter((row) => rowMatchesSearch(row, searchQuery,
searchCriterion));
  if (!sortKey) return filtered;
  return [...filtered].sort((a, b) => {
    const c = compareSortValues(a, b, sortKey);
    return sortDir === 'asc' ? c : -c;
  });
}, [allOrders, searchQuery, searchCriterion, sortKey, sortDir]);

const toggleSort = (key) => {
  if (sortKey === key) {
    setSortDir((d) => (d === 'asc' ? 'desc' : 'asc'));
  } else {
    setSortKey(key);
    setSortDir('asc');
  }
};

const fetchOrders = useCallback(async () => {
  try {
    setListLoading(true);
    setListError("");
    const response = await ordersViewAPI.getAll();
    const parsed = parseApiResponse(response);
    if (!parsed.ok) {
      setListError(parsed.message);
      setAllOrders([]);
      return;
    }
    setAllOrders(Array.isArray(parsed.data) ? parsed.data : []);
  } catch (err) {
    const parsed = parseApiResponse(err);
    setListError(parsed.message);
  }
};

```

```

    setAllOrders([]);
  } finally {
    setListLoading(false);
  }
}, []);

useEffect(() => {
  fetchOrders();
}, [fetchOrders]);

useEffect(() => {
  let cancelled = false;
  (async () => {
    setLookupsLoading(true);
    setClientsLookupError("");
    setEmployersLookupError("");
    try {
      const [cRes, eRes] = await Promise.all([
        clientsViewAPI.getAll(),
        employersViewAPI.getAll(),
      ]);
      const cParsed = parseApiResponse(cRes);
      const eParsed = parseApiResponse(eRes);
      if (cancelled) return;
      if (!cParsed.ok) {
        setClientsLookupError(cParsed.message);
        setClientsRows([]);
      } else {
        setClientsRows(Array.isArray(cParsed.data) ? cParsed.data : []);
      }
      if (!eParsed.ok) {
        setEmployersLookupError(eParsed.message);
        setEmployersRows([]);
      } else {
        setEmployersRows(Array.isArray(eParsed.data) ? eParsed.data : []);
      }
    } catch (err) {
      const p = parseApiResponse(err);
      if (!cancelled) {
        setClientsLookupError(p.message);
        setEmployersLookupError(p.message);
        setClientsRows([]);
        setEmployersRows([]);
      }
    }
  }) finally {

```

```

    if (!cancelled) setLookupsLoading(false);
  }
  })();
  return () => {
    cancelled = true;
  };
}, []);

const handleNewOrderChange = (field, value) => {
  setNewOrder((prev) => ({ ...prev, [field]: value }));
};

const handleCreateOrder = async (e) => {
  e.preventDefault();
  setFormError("");
  setFormSuccess("");

  const idClient = Number(newOrder.id_client);
  if (!Number.isInteger(idClient) || idClient <= 0) {
    setFormError('Оберіть клієнта зі списку. ');
    return;
  }
  if (!newOrder.name_device.trim()) {
    setFormError('Вкажіть назву / опис пристрою. ');
    return;
  }

  const descriptionTrimmed = newOrder.description.trim();
  if (descriptionTrimmed.length > 1000) {
    setFormError('Опис не може бути довшим за 1000 символів. ');
    return;
  }

  let idEmp = null;
  const empRaw = newOrder.id_emp.trim();
  if (empRaw) {
    const n = Number(empRaw);
    if (!Number.isInteger(n) || n <= 0) {
      setFormError('Некоректний ID працівника. ');
      return;
    }
    idEmp = n;
  }
}

```

```

const priceOrder = Math.max(0, Math.trunc(Number(newOrder.price_order) ||
0));

const payload = {
  id_client: idClient,
  name_device: newOrder.name_device.trim(),
  description: descriptionTrimmed ? descriptionTrimmed : null,
  id_emp: idEmp,
  date_creation: new Date().toISOString(),
  date_end: null,
  price_order: priceOrder,
  status_order: newOrder.status_order,
};

try {
  setCreating(true);
  const response = await ordersAPI.create(payload);
  const parsed = parseApiResponse(response);
  if (!parsed.ok) {
    const extra =
      parsed.details && parsed.details.length > 0
        ? ` ${parsed.details.join(' ')} `
        : '';
    setFormError(`${parsed.message} ${extra}`);
    return;
  }
  setFormSuccess('Замовлення успішно створено.');
  setNewOrder(emptyNewOrderForm());
  setShowNewOrderModal(false);
  await fetchOrders();
  setTimeout(() => setFormSuccess(''), 4000);
} catch (err) {
  const parsed = parseApiResponse(err);
  const extra =
    parsed.details && parsed.details.length > 0 ? ` ${parsed.details.join(' ')} ` : '';
  setFormError(`${parsed.message} ${extra}`);
} finally {
  setCreating(false);
}
};

const handleEditOrderChange = (field, value) => {
  setEditOrder((prev) => ({ ...prev, [field]: value }));
};

```

```

const openEditOrderModal = (row) => {
  if (row.id == null) return;
  setEditingOrderId(row.id);
  setEditingIdOrder(row.id_order);
  setEditOrder({
    id_client: String(row.id_client ?? ''),
    id_emp:
      row.id_emp != null && row.id_emp !== ''
      ? String(row.id_emp)
      : '',
    name_device: row.name_device ?? '',
    description: row.description ?? '',
    price_order: String(row.price_order ?? 0),
    status_order:
      row.status_order &&
      ORDER_STATUS_OPTIONS.includes(row.status_order)
      ? row.status_order
      : 'Прийнято',
  });
  setEditFormError('');
  setShowEditOrderModal(true);
};

const handleUpdateOrder = async (e) => {
  e.preventDefault();
  setEditFormError('');
  setFormSuccess('');

  if (editingOrderId == null) return;

  const idClient = Number(editOrder.id_client);
  if (!Number.isInteger(idClient) || idClient <= 0) {
    setEditFormError('Оберіть клієнта зі списку. ');
    return;
  }
  if (!editOrder.name_device.trim()) {
    setEditFormError('Вкажіть назву/ опис пристрою. ');
    return;
  }

  const descriptionTrimmed = editOrder.description.trim();
  if (descriptionTrimmed.length > 1000) {
    setEditFormError('Опис не може бути довшим за 1000 символів. ');
    return;
  }
}

```

```

let idEmp = null;
const empRaw = editOrder.id_emp.trim();
if (empRaw) {
  const n = Number(empRaw);
  if (!Number.isInteger(n) || n <= 0) {
    setEditFormError('Некоректний ID працівника.');
```

return;

```
  }
  idEmp = n;
}

const priceOrder = Math.max(0, Math.trunc(Number(editOrder.price_order) ||
0));

const payload = {
  id_client: idClient,
  name_device: editOrder.name_device.trim(),
  description: descriptionTrimmed ? descriptionTrimmed : null,
  id_emp: idEmp,
  price_order: priceOrder,
  status_order: editOrder.status_order,
};

try {
  setUpdating(true);
  const response = await ordersAPI.patch(editingOrderId, payload);
  const parsed = parseApiResponse(response);
  if (!parsed.ok) {
    const extra =
      parsed.details && parsed.details.length > 0
        ? ` ${parsed.details.join(' ')} `
        : "";
    setEditFormError(`${parsed.message} ${extra}`);
    return;
  }
  setFormSuccess('Замовлення успішно оновлено.');
```

closeEditOrderModal();

```
  await fetchOrders();
  setTimeout(() => setFormSuccess(""), 4000);
} catch (err) {
  const parsed = parseApiResponse(err);
  const extra =
    parsed.details && parsed.details.length > 0 ? ` ${parsed.details.join(' ')} ` : "";
  setEditFormError(`${parsed.message} ${extra}`);
} finally {
```

```

    setUpdating(false);
  }
};

const handleConfirmDeleteOrder = async () => {
  if (!deleteTarget?.id) return;
  setDeleteError("");
  setFormSuccess("");
  try {
    setDeleting(true);
    const response = await ordersAPI.remove(deleteTarget.id);
    const parsed = parseApiResponse(response);
    if (!parsed.ok) {
      setDeleteError(parsed.message);
      return;
    }
    setDeleteTarget(null);
    setFormSuccess('Замовлення видалено. ');
    await fetchOrders();
    setTimeout(() => setFormSuccess(""), 4000);
  } catch (err) {
    const parsed = parseApiResponse(err);
    setDeleteError(parsed.message);
  } finally {
    setDeleting(false);
  }
};

return (
  <Container fluid className="p-4 pb-0">
    <Row className="ps-5 align-items-center flex-wrap g-2">
      <Col xs={12} md>
        <h1 className="display-6 fw-bold text-primary mb-1">Замовлення</h1>
        <p className="text-muted mb-0">Список усіх замовлень</p>
      </Col>
      <Col xs={12} md="auto" className="ps-md-0">
        <Button variant="primary" type="button" onClick={() =>
setShowNewOrderModal(true)}>
          Додати
        </Button>
      </Col>
    </Row>

    {listError && (
      <Alert variant="danger" className="app-page-flash-alert py-2">

```

```

    {listError}
  </Alert>
)}

{formSuccess && (
  <Alert variant="success" className="app-page-flash-alert py-2">
    {formSuccess}
  </Alert>
)}

<Modal
  show={showNewOrderModal}
  onHide={closeNewOrderModal}
  size="lg"
  centered
  scrollable
  backdrop={creating ? 'static' : true}
  keyboard={!creating}
>
  <Modal.Header closeButton={!creating}>
    <Modal.Title>Нове замовлення</Modal.Title>
  </Modal.Header>
  <Modal.Body>
    {formError && (
      <Alert variant="danger" className="py-2">
        {formError}
      </Alert>
    )}
    <Form id="orders-new-order-form" onSubmit={handleCreateOrder}>
      <OrderFormFields
        fieldPrefix="new-order"
        values={newOrder}
        onFieldChange={handleNewOrderChange}
        clientsRows={clientsRows}
        employersRows={employersRows}
        lookupsLoading={lookupsLoading}
        clientsLookupError={clientsLookupError}
        employersLookupError={employersLookupError}
        disabled={creating}
      />
    </Form>
  </Modal.Body>
  <Modal.Footer>
    <Button variant="secondary" type="button" disabled={creating}
      onClick={closeNewOrderModal}>

```

```

    Скасувати
  </Button>
  <Button
    variant="primary"
    type="submit"
    form="orders-new-order-form"
    disabled={creating}
  >
    {creating ? (
      <>
        <Spinner animation="border" size="sm" className="me-2" />
        Створення...
      </>
    ) : (
      'Створити замовлення'
    )}
  </Button>
</Modal.Footer>
</Modal>

<Modal
  show={showEditOrderModal}
  onHide={closeEditOrderModal}
  size="lg"
  centered
  scrollable
  backdrop={updating ? 'static' : true}
  keyboard={!updating}
>
  <Modal.Header closeButton={!updating}>
    <Modal.Title>
      Редагування замовлення
      {editingIdOrder !== null ? ` № ${editingIdOrder}` : ""}
    </Modal.Title>
  </Modal.Header>
  <Modal.Body>
    {editFormError && (
      <Alert variant="danger" className="py-2">
        {editFormError}
      </Alert>
    )}
    <Form id="orders-edit-order-form" onSubmit={handleUpdateOrder}>
      <OrderFormFields
        fieldPrefix="edit-order"
        values={editOrder}

```

```

        onChange={handleEditOrderChange}
        clientsRows={clientsRows}
        employersRows={employersRows}
        lookupsLoading={lookupsLoading}
        clientsLookupError={clientsLookupError}
        employersLookupError={employersLookupError}
        disabled={updating}
      />
    </Form>
  </Modal.Body>
  <Modal.Footer>
    <Button variant="secondary" type="button" disabled={updating}
onClick={closeEditOrderModal}>
      Скасувати
    </Button>
    <Button
      variant="primary"
      type="submit"
      form="orders-edit-order-form"
      disabled={updating}
    >
      {updating ? (
        <>
          <Spinner animation="border" size="sm" className="me-2" />
          Збереження...
        </>
      ) : (
        'Зберегти зміни'
      )}
    </Button>
  </Modal.Footer>
</Modal>

<Modal
  show={Boolean(deleteTarget)}
  onHide={() => {
    if (deleting) return;
    closeDeleteModal();
  }}
  centered
  backdrop={deleting ? 'static' : true}
  keyboard={!deleting}
>
  <Modal.Header closeButton={!deleting}>
    <Modal.Title>Видалити замовлення?</Modal.Title>

```

```

</Modal.Header>
<Modal.Body>
  {deleteError && (
    <Alert variant="danger" className="py-2">
      {deleteError}
    </Alert>
  )}
  <p className="mb-0">
    Цю дію не можна скасувати. Видалити замовлення №{' '}
    <strong>{deleteTarget?.id_order}</strong>
    {deleteTarget?.name_device ? (
      <>
        {' '}
        – <span className="text-break">{deleteTarget.name_device}</span>
      </>
    ) : null}
    ?
  </p>
</Modal.Body>
<Modal.Footer>
  <Button variant="secondary" type="button" disabled={deleting}
onClick={closeDeleteModal}>
    Скасувати
  </Button>
  <Button variant="danger" type="button" disabled={deleting}
onClick={handleConfirmDeleteOrder}>
    {deleting ? (
      <>
        <Spinner animation="border" size="sm" className="me-2"/>
        Видалення...
      </>
    ) : (
      'Видалити'
    )}
  </Button>
</Modal.Footer>
</Modal>

<Row>
  <Col className="pt-4 pb-4" style={{ minWidth: 0 }}>
    <InputGroup className="mb-4">
      <Form.Control
        type="search"
        placeholder="Пошук..."
        value={searchQuery}

```

```

    onChange={(e) => setSearchQuery(e.target.value)}
    aria-label="Пошук замовлень"
  />
  <Form.Select
    value={searchCriterion}
    onChange={(e) => setSearchCriterion(e.target.value)}
    aria-label="Критерій пошуку"
    style={{ maxWidth: '220px' }}
  >
    {SEARCH_CRITERIA.map(({ value, label }) => (
      <option key={value} value={value}>
        {label}
      </option>
    ))}
  </Form.Select>
</InputGroup>

{listLoading ? (
  <div className="d-flex align-items-center justify-content-center py-5">
    <Spinner animation="border" role="status" variant="primary">
      <span className="visually-hidden">Завантаження...</span>
    </Spinner>
  </div>
) : (
  <div className="overflow-x-auto">
    <Table striped bordered hover className="mb-0 align-middle">
      <thead className="table-light">
        <tr>
          {TABLE_COLUMNS.map(({ sortKey: colKey, label }) => {
            const active = sortKey === colKey;
            return (
              <th
                key={colKey}
                scope="col"
                role="button"
                tabIndex={0}
                className="p-2 user-select-none"
                style={{
                  cursor: 'pointer',
                  whiteSpace: label.includes('\n')
                    ? 'pre-line'
                    : colKey === 'description'
                    ? 'normal'
                    : 'nowrap',
                  textAlign: label.includes('\n') ? 'center' : undefined,

```

```

        verticalAlign: 'middle',
        minWidth: colKey === 'description' ? '12rem' : undefined,
      }}
      onClick={() => toggleSort(colKey)}
      onKeyDown={(e) => {
        if (e.key === 'Enter' || e.key === ' ') {
          e.preventDefault();
          toggleSort(colKey);
        }
      }}
      aria-sort={
        active ? (sortDir === 'asc' ? 'ascending' : 'descending') : 'none'
      }
    >
    <span className="me-1">{label}</span>
    {active &&
      (sortDir === 'asc' ? (
        <ChevronUp className="d-inline" size={14} aria-hidden />
      ) : (
        <ChevronDown className="d-inline" size={14} aria-hidden />
      ))}
    </th>
  );
}}
<th className="p-2 text-center">Дії</th>
</tr>
</thead>
<tbody>
  {allOrders.length === 0 ? (
    <tr>
      <td colSpan={15} className="text-center text-muted py-4">
        Немає записів
      </td>
    </tr>
  ) : displayOrders.length === 0 ? (
    <tr>
      <td colSpan={15} className="text-center text-muted py-4">
        Нічого не знайдено за вашим запитом
      </td>
    </tr>
  ) : (
    displayOrders.map((row, index) => (
      <tr
        key={row.id ?? `${row.id_order}-${index}`}
        style={{

```

```

        cursor: row.id_order != null ? 'pointer' : undefined,
      }}
      onClick={e => {
        if (row.id_order == null) return;
        if (e.target.closest('button')) return;
        navigate(`/orders/${row.id_order}`);
      }}
    >
    <td className="p-2">{row.id_order}</td>
    <td className="p-2">{row.id_client != null ? row.id_client : '-'}
  </td>

    <td className="p-2">{row.name_client}</td>
    <td className="p-2">{row.phone_client}</td>
    <td className="p-2">{row.name_device}</td>
    <td className="p-2 text-break"
      style={{ maxWidth: '20rem', whiteSpace: 'pre-wrap' }}
    >
      {row.description != null && row.description !== ""
        ? row.description
        : '-'}
    </td>
    <td className="p-2">{row.id_emp != null ? row.id_emp : '-'}
  </td>

    <td className="p-2">{row.name_emp}</td>
    <td className="p-2">{formatDate(row.date_creation)}</td>
    <td className="p-2">{formatDate(row.estimated_date_end)}</td>
    <td className="p-2">{formatMoney(row.estimated_price)}</td>
    <td className="p-2">{formatDate(row.date_end)}</td>
    <td className="p-2">{formatMoney(row.price_order)}</td>
    <td className="p-2">{row.status_order}</td>
    <td className="p-2 text-center text-nowrap">
      <Button
        type="button"
        variant="outline-primary"
        size="sm"
        className="me-1 align-middle"
        disabled={row.id == null}
        onClick={e => {
          e.stopPropagation();
          openEditOrderModal(row);
        }}
        aria-label={`Редагувати замовлення ${row.id_order}`}
        title="Редагувати"
      >
      <PencilSquare size={16} aria-hidden />
    </td>
  </tr>
</tbody>
</table>

```

```

</Button>
<Button
  type="button"
  variant="outline-danger"
  size="sm"
  className="align-middle"
  disabled={row.id == null}
  onClick={(e) => {
    e.stopPropagation();
    setDeleteError("");
    setDeleteTarget({
      id: row.id,
      id_order: row.id_order,
      name_device: row.name_device ?? "",
    });
  }}
  aria-label={`Видалити замовлення ${row.id_order}`}
  title="Видалити"
>
  <Trash size={16} aria-hidden />
</Button>
</td>
</tr>
))
)}
</tbody>
</Table>
</div>
)}
</Col>
</Row>
</Container>
);
}

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Інженерія програмного забезпечення комп'ютерного сервісу

Виконав
студент групи КТН-112

Дмитро ГОГОЛЕНКО

Керівник
к. т. н., доцент

Євгеній ГОФМАН

2026

Рисунок В.1 – Слайд №1

Мета та завдання дослідження

□ Об'єкт дослідження

процес інженерії програмного забезпечення обліку замовлень, клієнтів, працівників, послуг і запасних частин у комп'ютерному сервісі.

❖ Предмет дослідження

методи та програмні засоби для розроблення інформаційних систем на базі архітектури клієнт-сервер.

➤ Мета роботи

підвищення рівня автоматизації та впорядкованості обліку роботи комп'ютерного сервісу шляхом розроблення вебзастосунку для ведення клієнтів, працівників, замовлень, послуг, запасних частин і пов'язаних операцій.

2

Рисунок В.2 – Слайд №2

Аналіз існуючих рішень

Порівняння з аналогами: RemOnline, HubSpot CRM, Google Таблиці

Основні недоліки існуючих рішень:

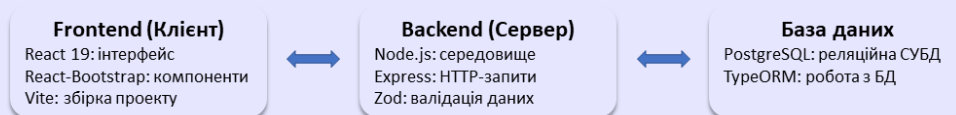
- абонентська плата або обмеження
- надлишковий функціонал
- слабка відповідність ремонтному циклу
- обмежене розмежування ролей і прав доступу

Критерій	RemOnline	HubSpot CRM	Google Таблиці	Сервісний центр (розроблюваний продукт)
Орієнтація на предметну область	Висока	Низька	Низька	Висока
Зв'язки між даними	Так	Обмежено	Частково	Так
Ролі доступу	Так	Так	Частково	Так
Витрати	Абонплата	Абонплата	Без витрат	Сервер
API для інтеграцій	Обмежено	Так	Відсутній	Так
Зведення даних	Обмежено	Так	Обмежено	Так

3

Рисунок В.3 – Слайд №3

Архітектура системи



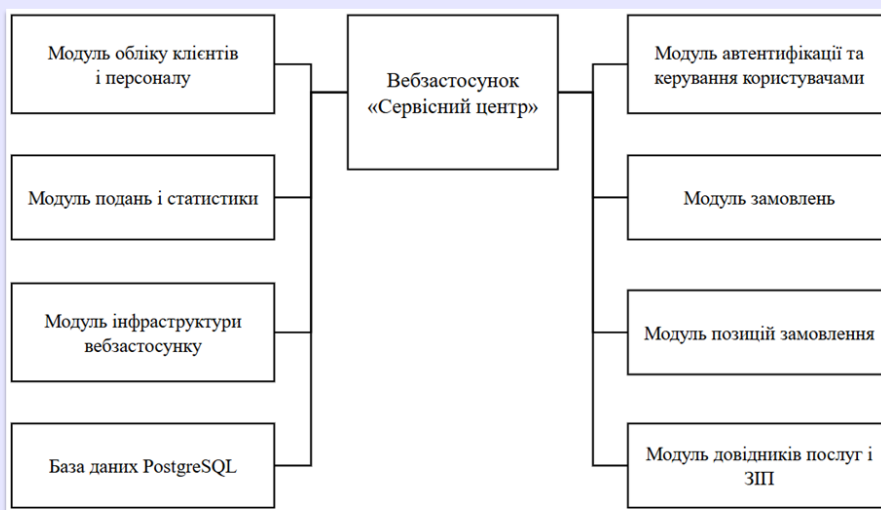
Порівняння підходів до реалізації

Критерій	JS (Node.js + React)	Python (Django + JS)	C# (ASP.NET Core + SPA)
Єдність мови клієнта й сервера	Висока	Низька	Низька або часткова
Екосистема пакетів	Дуже висока	Висока	Висока
Асинхронне I/O	Висока	Висока	Висока
Статична типізація	Низька	Низька	Висока
Поріг входу для розробки	Низький	Середній	Середній
Швидкість розробки інтерфейсу (UI)	Дуже висока	Середня	Середня
Швидкість розробки MVP	Висока	Середня	Низька

4

Рисунок В.4 – Слайд №4

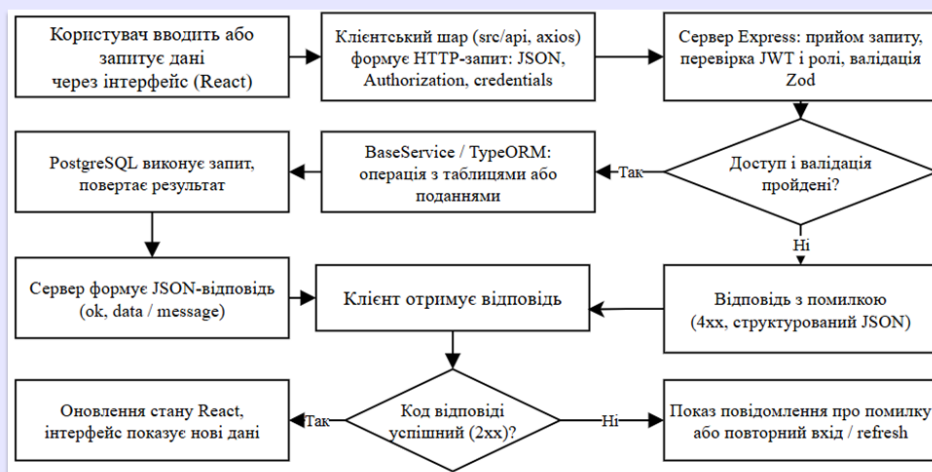
Структура системи



5

Рисунок В.5 – Слайд №5

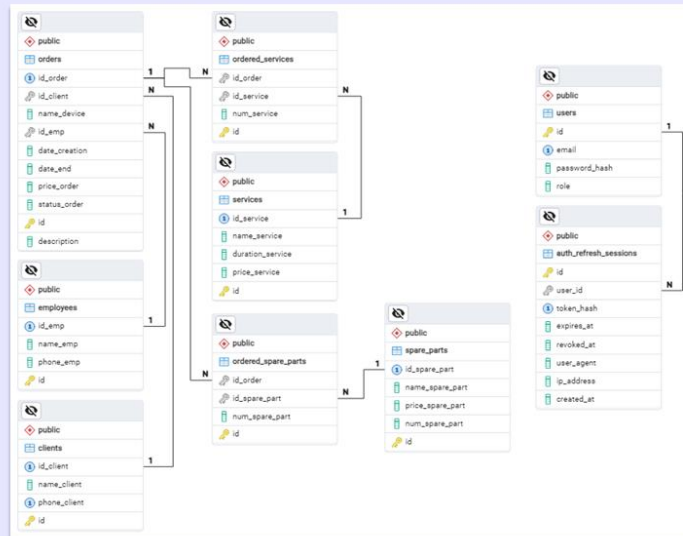
Обмін даними у системі



6

Рисунок В.6 – Слайд №6

Розробка бази даних



7

Рисунок В.7 – Слайд №7

Безпека та автентифікація

Заходи безпеки

- безпечне зберігання паролів
- токена автентифікація JWT
- керування refresh-сесіями
- рольова авторизація RBAC
- серверна перевірка прав
- валідація вхідних даних
- цілісність даних у PostgreSQL
- захист конфігурації та CORS

Вхід
Увійдіть у свій акаунт

Email
you@example.com

Пароль

Увійти

8

Рисунок В.8 – Слайд №8

Рольова модель

admin

- ✓ додавання облікових записів
- ✓ перегляд і видалення облікових записів
- ✓ права ролі app_user

app_user

- ✓ замовлення
- ✓ запчастини та послуги
- ✓ клієнти та працівники
- ✓ статистика

База даних

Роль зберігається в таблиці users

Сервер (JWT)

Валідація через middleware

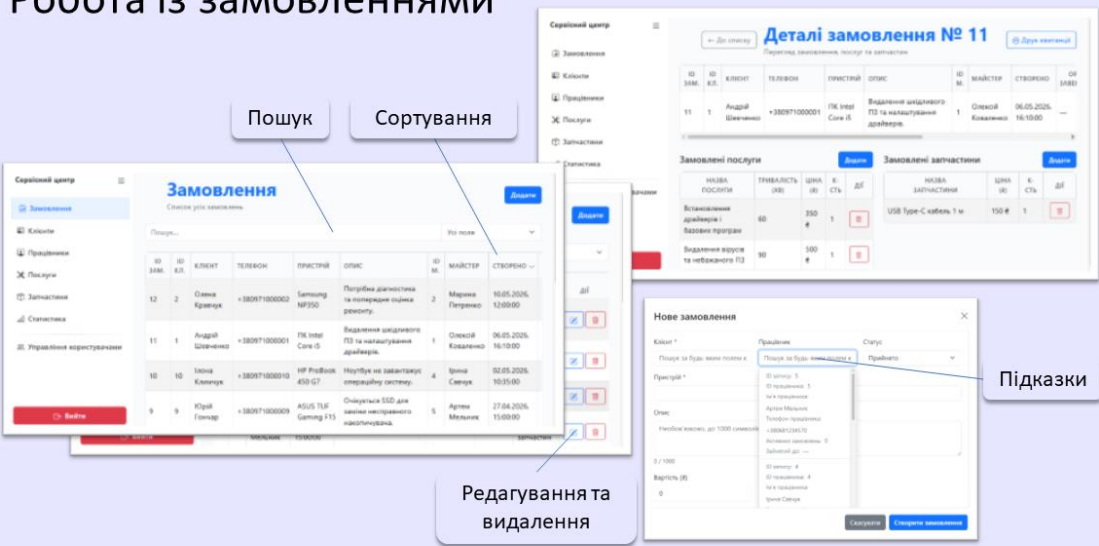
Клієнт

Динамічні маршрути та UI-елементи

9

Рисунок В.9 – Слайд №9

Робота із замовленнями



10

Рисунок В.10 – Слайд №10

Довідники і статистика

Запчастини

Довідник запчастин та залишок на складі

Додати

Пошук... Усі поля

ID	НАЗВА ЗАПЧАСТИНИ	ЦІНА (₴)	КІЛЬКІСТЬ НА СКЛАДІ	ДІЇ
12	Intel Core i5-10400F	4 200 ₴	1	
11	Блок живлення Chiefes 650W	2 800 ₴	3	

Послуги

Довідник послуг сервісного центру

Додати

Пошук... Усі поля

ID ПОСЛ.	НАЗВА ПОСЛУГИ	ТРИВАЛІСТЬ (ХВ)	ЦІНА (₴)	ДІЇ
1	Діагностика комп'ютера або ноутбука	60	300 ₴	
2	Чистлення системного блоку від пилу	45	350 ₴	

Клієнти

Облік клієнтів сервісного центру

Додати

Пошук... Усі поля

ID КЛ.	КЛІЄНТ	ТЕЛЕФОН	ВСЬОГО ВИТРАЧЕНО	Є ЗНИЖКА	ДІЇ
1	Андрій Шевченко	+380971000001	2 920 ₴	Так	
2	Олена Кравчук	+380971000002	1 250 ₴	Ні	

Працівники

Облік майстрів та інших працівників

Додати

Пошук... Усі поля

ID ПР.	ПРАЦІВНИК	ТЕЛЕФОН	АКТИВНИХ ЗАМОВЛЕНЬ	ЗАЙНЯТИЙ ДО	ДІЇ
1	Олексій Коваленко	+380671112233	0	—	
2	Марина Петренко	+380501234567	2	10.05.2026. 13:00:00	

Статистика

Місячні показники

Пошук... Усі поля

РІК	МІС.	ПРИЙНЯТО (ПОЧАТОК)	ОЧІКУВАННЯ З/Ч	ЗАВЕРШЕНО (ПОЧАТОК)	ОСКОРОВАНО (ПОЧАТОК)	ДОП. ЗАМОВЛЕННЯ (₴)	СЕРЕДНЬ ТРИВАЛІСТЬ (ДНІ)	НОВІ КЛІЄНТИ	ПОВТОРНІ КЛІЄНТИ
2 026	5	2	0	1	0	1 370 ₴	1	1	2
2 026	4	1	1	1	0	2 850 ₴	1	3	0

11

Рисунок В.11 – Слайд №11

Результати тестування

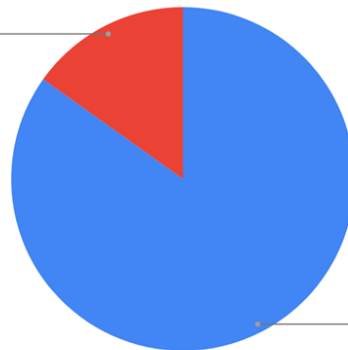
33 успішні тест-кейси. 100% покриття критичних сценаріїв

< 100 мс середній час відповіді API

20+ одночасних синхронних підключень

Успішність модульного тестування

Помилки
15,2%



Успішно
84,8%

12

Рисунок В.12 – Слайд №12

Висновки та перспективи

✓ Створено веб-застосунок
«Сервісний центр»

✓ Централізовано дані

✓ Реалізовано рольовий
доступ та статистику

○ Адаптація під малі екрани
та додавання темної теми

○ Розширений пошук,
фільтри та сповіщення

○ Підтримка інтеграції з
зовнішніми системами

Рисунок В.13 – Слайд №13