МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ СЕРВІСНОГО ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ»**

Виконав: студент 2 курсу, групи 8.1214-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Е.Р. Руденко
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Руденку Едуарду Руслановичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку для сервісного обслуговування автомобілів

керівник роботи Мухін Віталій Вікторович, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 326-с

2. Строк подання студентом роботи 05.12.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Технічне завдання.
2. Проєктування.
3. Програмна реалізація та тестування вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	01.04.2025	
2.	Збір вихідних даних.	16.04.2025	
3.	Обробка методичних та теоретичних	01.05.2025	
	джерел.		
4.	Розробка першого та другого розділу.	01.08.2025	
5.	Розробка третього розділу.	17.11.2025	
6.	Оформлення та нормоконтроль	01.12.2025	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	19.12.2025	

Студент _____ Е.Р. Руденко
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебзастосунку для сервісного обслуговування автомобілів»: 77 с., 44 рис., 14 джерел, 2 додатка.

АВТОСЕРВІС, ВЕБДОДАТОК, ВЕБРОЗРОБКА, ЗАМОВЛЕННЯ, ІНФОРМАЦІЙНА СИСТЕМА, КОРИСТУВАЧ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, РЕЄСТРАЦІЯ, ФРОНТЕНД, HTML, MONGODB, NODE.JS, REACT, TYPESCRIPT.

Об’єкт дослідження – процес проєктування інформаційних систем на основі React.

Предмет дослідження – технології створення вебдодатків.

Мета роботи – дослідження та розробка вебдодатку з використанням TypeScript та React.

Методи дослідження – аналіз аналогів, порівняльний аналіз функціональності, об’єктно-орієнтоване проєктування, клієнт-серверна архітектура, тестування.

У роботі розглянуто сучасні принципи створення вебресурсів для автосервісів, проведено аналіз існуючих рішень та описано інструменти й технології (HTML, CSS, TypeScript, React, Node.js, MongoDB), що використані для розробки вебдодатку з функціями реєстрації користувачів, обробки замовлень та керування інформацією; здійснено тестування працездатності та економічне обґрунтування. Результатом є вебзастосунок, який поєднує сучасні технології фронтенду й бекенду, інтегрується з MongoDB та реалізує клієнт-серверну архітектуру, узгоджуючись із сучасними підходами до розробки інформаційних систем у сфері послуг. Рекомендовано впровадження у малих та середніх автосервісах для автоматизації процесів, а значимість роботи визначається зростаючим попитом на цифрові інструменти управління; подальший розвиток передбачає розширення функціональності, аналітику та мобільний доступ.

SUMMARY

Master’s qualifying paper “Development of a Web Application for Car Service Maintenance”: 77 pages, 44 figures, 14 references, 2 supplements.

CAR SERVICE, WEB APPLICATION, WEB DEVELOPMENT, ORDERS, INFORMATION SYSTEM, USER, SOFTWARE, REGISTRATION, FRONTEND, HTML, MONGODB, NODE.JS, REACT, TYPESCRIPT.

The object of the study is the process of designing information systems based on the React framework.

The subject of the study is the technologies for creating web applications.

The aim of the study is the research and development of a web application using TypeScript and React.

The methods of research are analysis of analogues, comparative analysis of functionality, object-oriented design, client-server architecture, and application testing.

In the work, modern principles of creating web resources for car services are considered, an analysis of existing solutions is carried out, and the tools and technologies (HTML, CSS, TypeScript, React, Node.js, MongoDB) used for the development of a web application with functions of user registration, order processing, and information management are described; functionality testing and economic justification have been performed. The result is a web application that combines modern frontend and backend technologies, integrates with MongoDB, and implements a client-server architecture, corresponding to modern approaches to the development of information systems in the service sector. Implementation in small and medium-sized car services is recommended for the automation of processes, and the significance of the work is determined by the growing demand for digital management tools; further development involves the expansion of functionality, analytics, and mobile access.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Технічне завдання на виконаний програмний продукт 11
1.1 Терміни та визначення..... 11
1.1.1 Загальні терміни..... 11
1.1.2 Технічні терміни 11
1.2 Функціональні вимоги..... 12
1.3 Нефункціональні вимоги..... 13
1.3.1 Інтерфейс користувача 13
1.3.2 Підтримка браузерів 13
1.3.3 Вимоги до продуктивності..... 14
1.3.4 Вимоги до безпеки 14
1.4 Опис предметної області 15
1.5 Опис системи 16
1.6 Порівняння з подібними системами..... 17
2 Проєктування..... 26
2.1 Використання UML під час розробки системи 26
2.2 Діаграма варіантів використання 26
2.2.1 Опис варіантів використання..... 30
2.3 Діаграма класів для прецеденту «Керування замовленнями»..... 33
2.4 Діаграма послідовності для прецеденту«Керування замовленнями» 34
2.5 Діаграма розгортання для системи автосервісу..... 36
3 Реалізація та тестування 38
3.1 Опис інструментів розробки 38
3.2 Основні компоненти системи 39

3.2.1 Основні функціональні модулі фронтенд-частини 40
3.2.2 Основні функціональні модулі бекенд-частини 43
3.2.3 Схема роботи бази даних 48
3.3 Тестування проєкту 51
3.3.1 Модульне тестування (Unit-тестування) 60
3.3.2 Інтеграційне тестування 64
3.4 Керівництво користувача 65
3.4.1 Рівень підготовки користувача 65
3.4.2 Підготовка до роботи 65
3.4.3 Вхід до системи 67
Висновки 70
Перелік посилань 72
Додаток А Код тестування EditInvoiceForm.tsx 74
Додаток Б Код тестування CreateOrder.tsx 77

ВСТУП

У сучасних умовах цифрової трансформації бізнес-процесів особливого значення набуває автоматизація діяльності підприємств сфери обслуговування. Автомобільний сервіс як один із найбільш затребуваних сегментів ринку технічного обслуговування потребує впровадження ефективних інформаційних систем, здатних забезпечити оптимізацію роботи персоналу, підвищення якості послуг та скорочення часу виконання ремонтних операцій. Зростання кількості автомобілів в Україні та у світі зумовлює збільшення навантаження на автосервісні центри, що висуває нові вимоги до швидкості, точності та прозорості організації роботи. За цих умов актуальною стає проблема цифровізації ключових бізнес-процесів автосервісів.

Традиційні методи ведення обліку замовлень, планування ремонтних робіт і комунікації з клієнтами часто є малоефективними, не масштабуються та супроводжуються високими ризиками виникнення помилок. Багато невеликих та середніх автосервісів усе ще використовують паперові картки, базові електронні таблиці або застарілі локальні програми, що не дозволяють оперативно контролювати процеси, аналізувати показники ефективності та забезпечувати якісний клієнтський сервіс. У зв’язку з цим виникає потреба у впровадженні сучасних веборієнтованих інформаційних систем, побудованих на гнучких архітектурах, здатних обробляти значні обсяги даних у режимі реального часу.

Розробка вебзастосунку для управління процесами автосервісу є важливим завданням, оскільки дозволяє автоматизувати рутинні операції прийом замовлень, облік виконаних робіт, формування документів, відстеження статусів ремонту, взаємодію персоналу та ведення бази клієнтів. Такі системи сприяють підвищенню продуктивності роботи майстрів, мінімізації людського фактора, підсиленню прозорості бізнес-процесів та підвищенню задоволеності клієнтів.

У межах магістерської роботи розглядається створення веборієнтованої інформаційної системи, яка спрямована на вирішення проблеми неефективного управління обслуговуванням автомобілів на підприємствах сервісного типу. Система розробляється з урахуванням вимог сучасних технологій програмної інженерії, принципів побудови безпечних вебзастосунків, можливості масштабування та гнучкого налаштування під потреби різних автосервісних центрів.

Метою дослідження є розроблення та обґрунтування архітектури веборієнтованої інформаційної системи для автоматизації процесів автосервісу, а також створення програмного забезпечення, що забезпечує ефективне управління замовленнями, персоналом і процесами технічного обслуговування автомобілів.

Для досягнення поставленої мети необхідно виконати такі завдання:

– проаналізувати сучасний стан автоматизації автосервісів та існуючі інформаційні системи цього сегмента;
– дослідити вимоги до вебсистем обліку та роботи сервісних підприємств;
– розробити концептуальну модель інформаційної системи та визначити ключові бізнес-процеси;
– створити UML-діаграми, що описують структуру та поведінку системи;
– обґрунтувати вибір технологій (React, Node.js, MongoDB тощо) та архітектурних рішень;
– реалізувати програмну частину вебзастосунку з підтримкою авторизації, CRUD-операцій, управління замовленнями та генерації звітів;
– провести тестування системи та оцінити її ефективність;
– визначити практичну значущість і перспективи впровадження розробленого рішення.

Об’єктом дослідження є процеси управління технічним обслуговуванням та ремонтом автомобілів у автосервісних центрах.

Предметом дослідження є методи, моделі та програмні засоби автоматизації бізнес-процесів автосервісу, реалізовані у вигляді веборієнтованої інформаційної системи.

У ході роботи застосовано такі методи:

– аналіз існуючих інформаційних систем у сфері автосервісу;
– методи структурного та об’єктно-орієнтованого моделювання (UML);
– порівняльний аналіз технологій фронтенд- та бекенд-розробки;
– методи проєктування баз даних;
– методи програмної реалізації вебзастосунків;
– експериментальні методи тестування функціональності системи;
– аналіз ефективності обробки даних і взаємодії користувачів із системою.

Наукова новизна отриманих результатів полягає у:

– запропонованій архітектурі вебсистеми для автосервісів, що поєднує асинхронну обробку даних, модульну логіку та гнучке керування бізнес-процесами;
– розробленій моделі оптимізації процесу обробки замовлень на основі ролей користувачів (клієнт, персонал, менеджер);
– удосконаленні підходів до автоматизації сервісних процесів шляхом застосування сучасних вебтехнологій (React + Node.js + MongoDB);
– створенні універсального програмного модуля для формування електронних документів (рахунків, квитанцій).

Практичне значення полягає у створенні веборієнтованої інформаційної системи, яка може бути впроваджена в автосервісні центри будь-якого масштабу. Система дозволяє:

– автоматизувати обробку замовлень;
– скоротити час прийому клієнта;
– оптимізувати роботу персоналу;
– забезпечити зручну взаємодію клієнта з автосервісом;
– підвищити якість управління та облік виконаних робіт.

1 ТЕХНІЧНЕ ЗАВДАННЯ НА ВИКОНАНИЙ ПРОГРАМНИЙ ПРОДУКТ

1.1 Терміни та визначення

1.1.1 Загальні терміни

У цьому підпункті подано визначення основних термінів, що використовуються в роботі.

Інформаційна система – організована сукупність технічних засобів, програмного забезпечення, даних, процедур і персоналу, призначена для збору, зберігання, обробки та передачі інформації з метою підтримки прийняття рішень.

Вебдодаток – програмний продукт, що працює на вебсервері та доступний через веббраузер без необхідності встановлення на клієнтському пристрої.

Користувач – фізична або юридична особа, яка взаємодіє з програмним забезпеченням із певною метою.

Розробка програмного забезпечення – процес створення, проєктування, впровадження та підтримки програмних продуктів.

Автоматизація бізнес-процесів – впровадження технологій, що дозволяють зменшити участь людини у виконанні стандартних рутинних операцій у межах підприємницької діяльності.

1.1.2 Технічні терміни

У цьому підпункті подано визначення термінів, пов’язаних з використаними технологіями та інструментами.

React – JavaScript-бібліотека для побудови інтерфейсів користувача, орієнтована на створення односторінкових додатків (SPA).

TypeScript – мова програмування, яка є надмножиною JavaScript і додає до нього статичну типізацію.

Node.js – програмна платформа, що дозволяє запускати JavaScript-код на сервері.

MongoDB – документоорієнтована система управління базами даних, яка використовує формат JSON-подібних документів.

API (Application Programming Interface) – інтерфейс прикладного програмування, який дозволяє різним програмним модулям взаємодіяти між собою.

CRUD-операції – базові операції над даними: створення (Create), читання (Read), оновлення (Update) та видалення (Delete).

Frontend – частина вебдодатку, з якою безпосередньо взаємодіє користувач (інтерфейс).

Backend – серверна частина додатку, яка забезпечує логіку роботи, обробку запитів і взаємодію з базами даних.

1.2 Функціональні вимоги

Система має надавати користувачам такі можливості:

– перегляд загальної інформації про автосервіс;
– облік та редагування особистих даних клієнтів;
– перегляд статусу власних замовлень;
– взаємодія з менеджером (запити, callback, комунікації);
– перегляд публікацій, інформаційних повідомлень та новин.

Система має надавати менеджерам і майстрам такі можливості:

– створення/редагування/перегляд/видалення замовлень;
– ведення обліку робіт та матеріалів;

– контроль статусів ремонту та строків виконання;
– управління щоденними завданнями;
– робота з контактними зверненнями клієнтів;
– формування рахунків та супровідних документів.

Система має надавати адміністраторам такі можливості:

– створення, редагування та видалення користувачів;
– призначення ролей (адміністратор, менеджер, майстер, клієнт);
– управління публікаціями (створення новин, інформаційних сторінок);
– повний доступ до всіх CRUD-операцій;
– моніторинг коректності роботи системи.

1.3 Нефункціональні вимоги

1.3.1 Інтерфейс користувача

Система повинна забезпечувати:

– інтуїтивно зрозумілий інтерфейс з чіткою структурою навігації;
– адаптивний дизайн, що коректно відображає елементи інтерфейсу на пристроях різної роздільної здатності (ПК, ноутбуки, смартфони, планшети);
– підтримку світлої та темної теми (опціонально);
– відповідність сучасним стандартам UI/UX.

1.3.2 Підтримка браузерів

Система повинна стабільно працювати в останніх версіях таких браузерів:

– Google Chrome;

– Mozilla Firefox;
– Microsoft Edge;
– Safari (мобільні та десктопні версії).

Підтримка Internet Explorer не вимагається.

1.3.3 Вимоги до продуктивності

Час відображення будь-якої сторінки або виконання основної операції (створення, редагування, видалення, авторизація) не повинен перевищувати 2 секунд при номінальному навантаженні.

Система повинна обробляти не менше 1000 запитів на хвилину без критичного зниження швидкодії.

API має відповідати принципам REST та забезпечувати ефективну роботу в асинхронному режимі.

1.3.4 Вимоги до безпеки

Система повинна забезпечувати:

– автентифікацію та авторизацію на основі JWT з контролем терміну дії токена;
– шифрування всіх переданих даних через HTTPS;
– захист від основних типів атак, зокрема: SQL/NoSQL-injection, XSS (Cross-Site Scripting),CSRF (Cross-Site Request Forgery), brute-force атак на авторизацію;
– розмежування прав доступу між ролями користувачів;
– заборону доступу неавторизованим користувачам до будь-яких внутрішніх даних;
– окремий інтерфейс адміністратора з обмеженим доступом.

1.4 Опис предметної області

Предметною областю є розробка веборієнтованої інформаційної системи для автоматизації діяльності автосервісного підприємства. Така система повинна надавати користувачам можливість керування замовленнями, обліком клієнтів, фінансовими операціями, телефонними зверненнями, поточними роботами, а також адмініструванням користувачів та інформаційних матеріалів. Основною метою функціонування системи є оптимізація бізнес-процесів автосервісу, підвищення продуктивності праці персоналу та покращення якості обслуговування клієнтів.

Процес взаємодії користувача з системою відбувається у веббраузері. Для початку роботи користувач вводить адресу сайту та здійснює авторизацію, ввівши логін і пароль у форму входу. Якщо користувач отримав доступ уперше, необхідні облікові дані йому надає адміністратор системи. Після успішної аутентифікації користувач отримує доступ до функціональних модулів, що залежать від його ролі у системі (адміністратор, менеджер, майстер, клієнт).

Система містить низку модулів, що забезпечують повний цикл обслуговування автомобіля. Наприклад, у модулі «Замовлення» користувач може створити нове замовлення, заповнивши форму з інформацією про клієнта, транспортний засіб, заплановані роботи та необхідні матеріали. Після натискання кнопки «Зберегти» дані передаються на сервер і зберігаються у базі даних. Нове замовлення автоматично з’являється у списку, де його можна переглянути, відредагувати або видалити згідно з правами доступу користувача.

У модулі «Щоденні роботи» персонал автосервісу може переглядати актуальний перелік завдань на день, відмічати виконані роботи та контролювати стан ремонтів у реальному часі. Аналогічним чином здійснюється робота в модулях «Рахунки», «Користувачі», «Телефонні звернення» та «Публікації». Усі модулі мають однакові принципи взаємодії: перегляд, створення, редагування та видалення записів.

Система забезпечує адміністраторам можливість керування користувачами, налаштовування ролей, модерації контенту, контролю звернень клієнтів та формування звітів щодо виконаних робіт, фінансових операцій або завантаженості персоналу. Адміністратор також відповідає за створення інформаційних сторінок та новин, які відображаються для клієнтів та працівників сервісу.

Таким чином, предметна область включає процеси реєстрації, збереження та обробки даних, пов’язаних з обслуговуванням автомобілів, комунікаціями з клієнтами та внутрішньою організацією діяльності автосервісу. Вебзастосунок сприяє оптимізації процесів, мінімізації ручних операцій, підвищенню точності інформації та забезпеченню високого рівня сервісу.ми не втратимо цю інформацію.

1.5 Опис системи

Система для автоматизації роботи автосервісу є ключовим інструментом підвищення ефективності управління підприємством, яке надає послуги з технічного обслуговування та ремонту автомобілів. У традиційних умовах облік замовлень, клієнтської інформації, фінансових операцій та телефонних звернень здійснюється вручну, у паперовому вигляді або за допомогою несистематизованих електронних таблиць. Такий підхід призводить до численних проблем: відсутності уніфікації даних, ризику втрати інформації, труднощів із контролем виконання робіт та значних часових витрат персоналу.

У зв’язку з цим зростає потреба у впровадженні сучасних веборієнтованих інформаційних систем, які дозволяють централізовано керувати всіма аспектами діяльності автосервісу. Розроблювана система покликана забезпечити користувачів зручним та ефективним інструментом для автоматизації процесів прийому замовлень, обліку виконаних робіт, формування фінансових документів, ведення контактних звернень та адміністрування користувачів.

1.6 Порівняння з подібними системами

RemOnline. Програмний продукт RemOnline [1], приклад інтерфейсу якого представлений на рисунку 1.1, є хмарним рішенням для автоматизації бізнес-процесів у сфері обслуговування та ремонту, включаючи автосервіси. Сервіс надає набір інструментів для обліку замовлень, контролю залишків на складах, ведення клієнтської бази та управління фінансами.

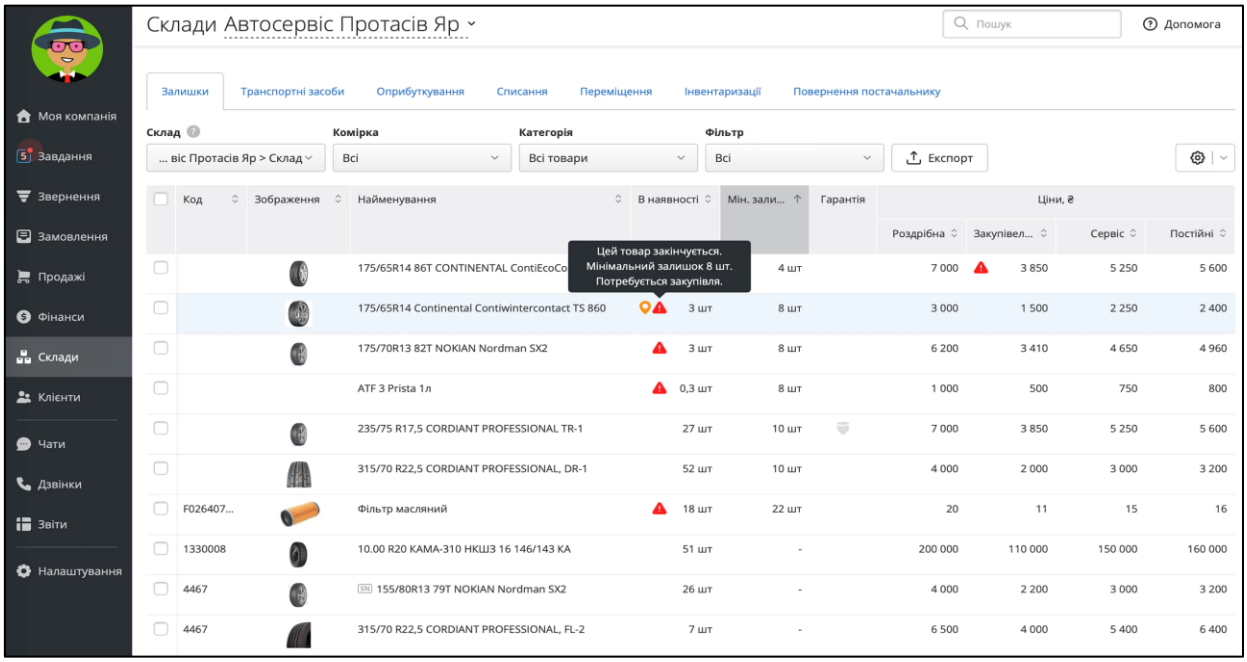


Рисунок 1.1 – Інтерфейс програми RemOnline (РемОнлайн) «Склади»

RemOnline позиціонується як рішення «все в одному» для малого та середнього бізнесу, що дозволяє відмовитися від використання декількох розрізнених сервісів.

Переваги:

– доступ до системи з будь-якої точки світу через вебінтерфейс або мобільний додаток;
– простота впровадження та відсутність потреби у встановленні програмного забезпечення на локальні машини;
– автоматизація процесів створення та обробки замовлень-нарядів;
– вбудована CRM-система для роботи з клієнтами;

– управління залишками, закупівлями та складським обліком;
– контроль грошових потоків, звітність за прибутком, витратами, касовими операціями;
– модуль аналітики, що дозволяє оцінювати ефективність працівників та загальний стан бізнесу;
– підтримка сповіщень для клієнтів через SMS/Viber;
– інтеграція з IP-телефонією, популярними платіжними сервісами та маркетплейсами.

Недоліки:

– щомісячна абонплата, яка при довготривалому використанні може перевищити вартість одноразової покупки ліцензії інших систем;
– повна залежність від стабільного інтернетз'єднання;
– обмеження на обсяг зберігання даних у залежності від тарифного плану;
– обмеження в кастомізації бізнеслогіки (неможливо глибоко адаптувати під специфічні бізнес-процеси без втручання розробників);
– деякі розширені функції доступні лише у дорогих тарифах.

Висновок: RemOnline є сучасним, гнучким і простим у використанні рішенням для автоматизації автосервісів. Його головна перевага швидкий старт і доступність з будь-якого пристрою. Проте довгострокове використання може бути фінансово обтяжливим через регулярну оплату підписки.

Ulteama. Програмний продукт Ulteama [2], приклад інтерфейсу якого представлений на рисунку 1.2 (стор. 20), є хмарною CRM-системою, спеціально розробленою для автоматизації бізнес-процесів малих та середніх автосервісів. Система охоплює ключові аспекти діяльності СТО: від планування робіт і ведення замовлень до фінансового обліку та управління персоналом.

Переваги:

– фінансовий облік: фіксація надходжень та витрат, звіти по руху грошових коштів, доступи до кас, статті оплат та витрат;

– планування та облік робіт: візуальний планувальник для розподілу завантаження постів та майстрів, індивідуальні графіки роботи, запис клієнтів на обслуговування;
– управління замовленнями: швидке створення замовлень, довідники послуг та запчастин, прикріплення фото та файлів до замовлення, розподіл замовлень між майстрами;
– складський облік: оприбуткування запчастин, списання на замовлення, повернення постачальнику, історія руху запчастин, контроль залишків;
– клієнтська база: ведення бази клієнтів, історія кожного клієнта, збереження інформації про автомобілі клієнтів, зв’язок клієнта з компаніями;
– звіти та аналітика: звіт прибутку по замовленнях, звіти по майстрам та послугам, звіти по руху грошових коштів, звіт по обороту клієнтів, додаткові звіти по замовленням;
– інтеграції: підбір запчастин через постачальників, інтеграція з телефонією Binotel, месенджерами;
– управління персоналом: контроль завантаженості співробітників, керування доступами, збереження історії змін, нарахування зарплат, бонуси та штрафи, детальні звіти по зарплатам;
– гнучке налаштування: налаштування бізнес-процесів та довідників, налаштування доступів та ролей, шаблони документів;
– технічна підтримка: допомога в налаштуваннях, імпорті даних, консультації з усіх питань.

Недоліки:

– вартість: щомісячна абонплата 49 євро за базовий пакет на 3 співробітників; кожен додатковий співробітник 5 євро, кожна додаткова локація 15 євро;
– залежність від інтернету: повна залежність від стабільного інтернет-з’єднання для доступу до системи;

– обмеження кастомізації можливості глибокої адаптації під специфічні бізнес-процеси можуть бути обмежені без втручання розробників;
– молодість продукту: порівняно новий продукт на ринку, що може впливати на наявність деяких функцій або інтеграцій.

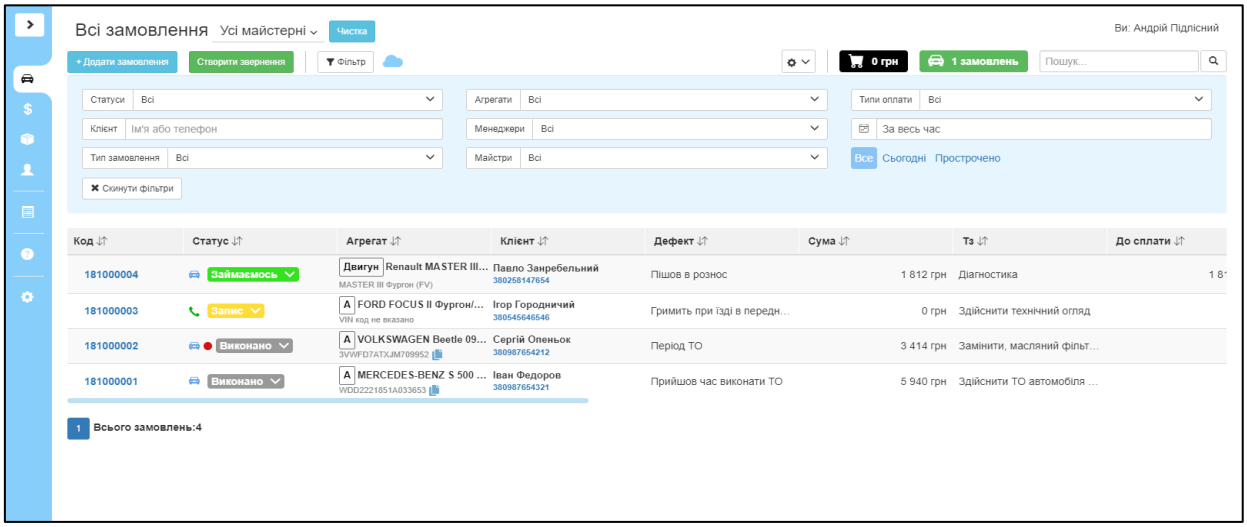


Рисунок 1.2 – Інтерфейс програми Ulteama «Замовлення-наряд»

Висновок: Ulteama сучасне хмарне рішення для автоматизації автосервісів, яке пропонує широкий спектр функцій для ефективного управління СТО. Його переваги включають зручний інтерфейс, комплексний підхід до обліку та управління, а також гнучкі можливості налаштування. Проте, варто враховувати щомісячну абонплату та залежність від інтернет-з'єднання при прийнятті рішення про впровадження системи.

Carbook Mobi. Програмний продукт Carbook Mobi [3], приклад інтерфейсу якого представлений на рисунку 1.3 (стор. 22), є хмарною CRM-системою, розробленою українською командою для автоматизації та управління процесами на станціях технічного обслуговування (СТО). Система охоплює ключові аспекти діяльності автосервісу: від запису клієнтів до фінансового обліку та аналітики.

Переваги:

– запис клієнтів: зручний інтерфейс для планування візитів та управління чергою;

– огляд автомобілів: фіксація стану авто при прийманні з можливістю додавання зауважень та друку акту приймання-передачі;
– замовлення запчастин через API: інтеграція з постачальниками для швидкого підбору та замовлення необхідних деталей;
– ремонт агрегатів: ведення історії ремонтів та обслуговування кожного автомобіля;
– задачі та нагадування: створення завдань для співробітників з автоматичними нагадуваннями;
– діагностика автомобілів: збереження результатів діагностики та рекомендацій для клієнтів;
– складський облік: управління залишками, списанням та оприбуткуванням запчастин;
– відгуки та узгодження з клієнтами: збір зворотного зв’язку та погодження робіт через систему;
– планування постів: оптимізація завантаження робочих місць та майстрів;
– сервісна книжка: ведення електронної історії обслуговування автомобіля;
– зарплати: автоматичний розрахунок заробітної плати та бонусів для співробітників;
– звіти та KPI: формування аналітичних звітів для оцінки ефективності роботи СТО;
– каса та РРО: інтеграція з фіскальними реєстраторами та облік готівкових операцій;
– власний сайт: можливість створення сайту СТО з онлайн-записом для клієнтів.

Недоліки:

– щомісячна абонплата: вартість використання програми починається від 700 грн на місяць, що може бути суттєвим для невеликих СТО;
– залежність від інтернету: повна функціональність програми доступна лише при стабільному інтернет-з’єднанні;

– обмеження кастомізації: можливості глибокої адаптації під специфічні бізнес-процеси можуть бути обмежені без втручання розробників;
– молодість продукту: порівняно новий продукт на ринку, що може впливати на наявність деяких функцій або інтеграцій.

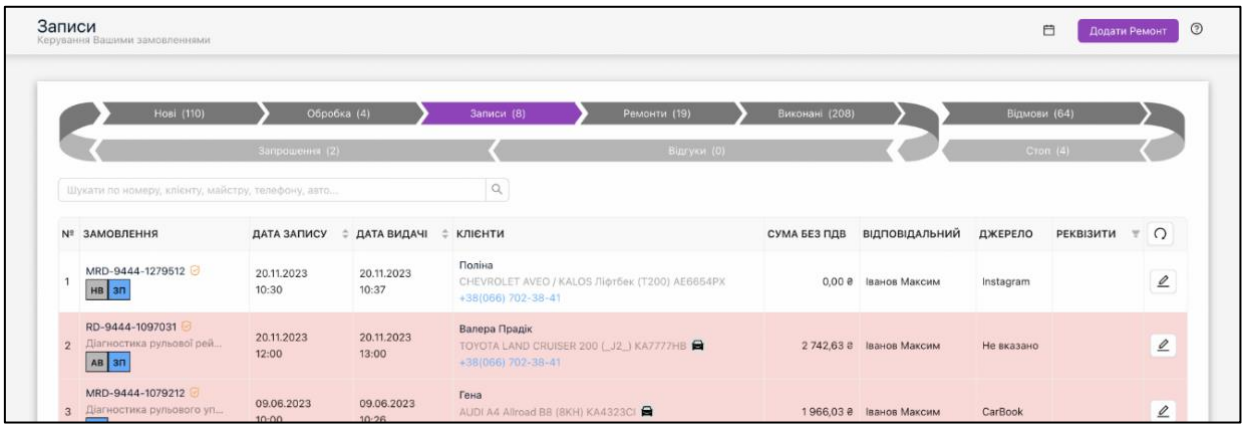


Рисунок 1.3 – Інтерфейс програми Carbook Mobi «Записи»

Висновок: Carbook Mobi – це сучасне хмарне рішення для автоматизації автосервісів, яке пропонує широкий спектр функцій для ефективного управління СТО. Його переваги включають зручний інтерфейс, комплексний підхід до обліку та управління, а також гнучкі можливості налаштування. Проте, варто враховувати щомісячну абонплату та залежність від інтернет-з'єднання при прийнятті рішення про впровадження системи.

Менеджер автосервісу (E-Soft Auto). Програмний продукт Менеджер автосервісу (E-Soft Auto) [4], приклад інтерфейсу якого представлено на рисунку 1.4, є спеціалізованим рішенням для автоматизації роботи станцій технічного обслуговування, автомагазинів та інших підприємств автобізнесу. Програма встановлюється на ПК та дозволяє організувати повний цикл обслуговування клієнтів – від попереднього запису до фінансового аналізу діяльності.

Переваги:

– облік клієнтів та їхніх авто: зберігання історії візитів, проведених робіт і рекламацій, що полегшує повторне обслуговування;

– повноцінне ведення замовлень-нарядів: можливість фіксації робіт, використаних запчастин, зауважень та рекомендацій, що дає змогу зручно контролювати процес обслуговування автомобіля;
– складський модуль: автоматизація приходу/витрат товарів, контроль мінімальних залишків, формування інвентаризаційних звітів;
– торговий модуль: можливість продажу автозапчастин із формуванням накладних, використанням штрих-кодів, друком цінників;
– фінансовий модуль: облік касових операцій, витрат, руху коштів, нарахування зарплат продавцям чи майстрам;
– гнучка система звітності: понад 20 типів звітів для аналізу діяльності;
– разова оплата ліцензії, передбачена купівля повної версії без щомісячної абонплати (вартість від $900).

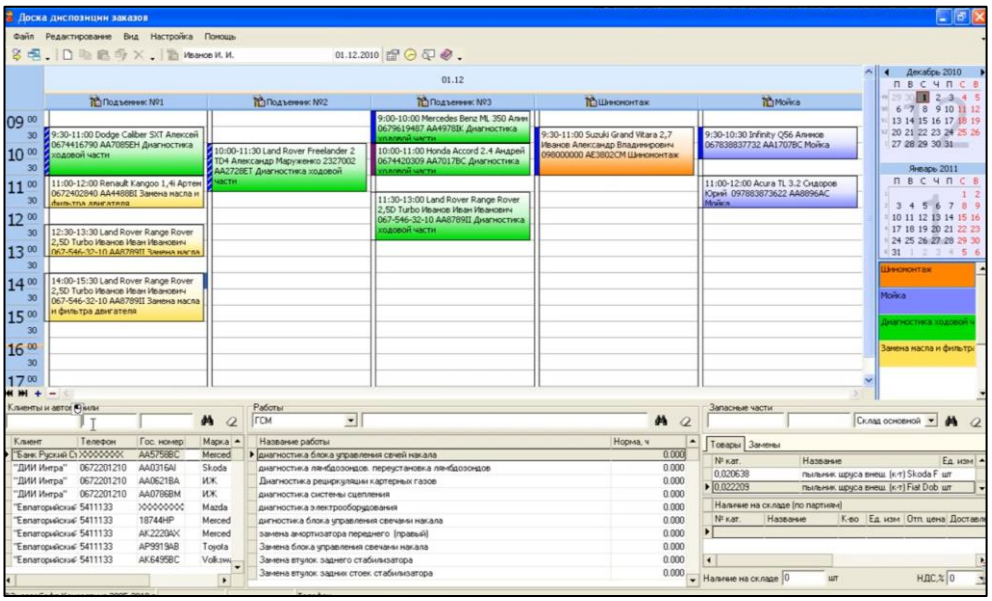


Рисунок 1.4 – Інтерфейс програми E-Soft Auto «Доска диспозиції нарядів»

Недоліки:

– застарілий інтерфейс: графічний інтерфейс програми виглядає застарілим і менш інтуїтивним у порівнянні з сучасними хмарними CRM-системами (це може впливати на швидкість адаптації нових користувачів);
– відсутність хмарного доступу;

- не підтримує інтеграції з IP-телефонією, Viber/Telegram;
- відсутність мобільного додатку або вебдоступу для майстрів: працівники не можуть переглядати свої завдання, оновлювати статуси замовлень або вести фотозвіт безпосередньо з майстерні;
- оновлення програми платні: нові версії або доопрацювання функціоналу на замовлення оплачуються окремо, що створює додаткові витрати при масштабуванні;
- обмежені можливості кастомізації під нестандартні бізнес-процеси;
- підтримка лише Windows: відсутня версія для macOS або Linux, що обмежує вибір операційних систем.

Висновок: E-Soft Auto – це стабільне та перевірене рішення для класичного автосервісу, яке добре підходить для підприємств, що віддають перевагу автономній роботі без залежності від інтернет-з'єднання. Основні функції (облік замовлень, склад, фінанси, торгівля) реалізовані на належному рівні. Проте, за відсутності хмарних технологій, інтеграцій, мобільного доступу та сучасного UX, система виглядає морально застарілою на фоні новітніх хмарних CRM-продуктів.

Узагальнений висновок. Порівняння з існуючими системами показує, що більшість сучасних рішень є хмарними, багатофункціональними та орієнтованими на середні й великі СТО. Вони надають широкі можливості, проте:

- вимагають регулярних фінансових витрат (абонплата);
- залежать від стабільного інтернетз'єднання;
- мають обмежені можливості кастомізації під специфічні процеси;
- не завжди оптимально підходять для невеликих або спеціалізованих автосервісів.

Розроблювана в рамках даної кваліфікаційної роботи система:

- може адаптуватися до індивідуальних бізнес-процесів;
- побудована на сучасних вебтехнологіях (React, Node.js, MongoDB);
- оптимізована під конкретні потреби автосервісу;

– не перевантажена зайвими модулями;

– має гнучку, легко доповнювану архітектуру.

Таким чином, розроблюваний вебзастосунок поєднує ключові переваги сучасних CRM-систем, але при цьому залишається доступним, гнучким і оптимізованим під потреби конкретного автосервісного підприємства.

Крім того, розроблена система демонструє здатність ефективно масштабуватися разом зі зростанням бізнесу. Завдяки модульній структурі та використанню MERN-стека, вона може бути розширена новими функціональними модулями – такими як управління складом запчастин, аналітика виконаних робіт, інтеграція з телефонією або підсистема лояльності клієнтів. Це робить застосунок перспективним для довгострокового використання і модернізації без необхідності радикальної зміни архітектури.

Система також вирізняється простотою розгортання та підтримки, що є важливою перевагою у порівнянні з комерційними SaaS-рішеннями. Вона може функціонувати як у локальній мережі невеликого автосервісу, так і в хмарному середовищі, забезпечуючи гнучкість вибору інфраструктури. Завдяки відкритому коду та відсутності прив’язки до сторонніх сервісів, підприємство отримує повний контроль над даними, що значно підвищує рівень безпеки та конфіденційності.

2 ПРОЄКТУВАННЯ

2.1 Використання UML під час розробки системи

Використання UML (Unified Modeling Language) під час розробки системи є ключовим елементом для успішного проєктування та впровадження програмного забезпечення. UML надає стандартизований набір графічних засобів та нотацій, які допомагають команді розробників зрозуміти, визначити та моделювати систему перед її реалізацією.

Уміння користуватись UML також сприяє покращенню комунікації між командою розробників та замовником. Графічне представлення системи у вигляді діаграм дозволяє замовнику краще зрозуміти, яким чином система буде функціонувати та які можливості вона надає. Це сприяє виявленню можливих розбіжностей між очікуваннями замовника та побудованою системою ще на ранніх етапах розробки.

2.2 Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) є одним із ключових інструментів методології UML (Unified Modeling Language), що використовується для моделювання взаємодії користувачів із програмною системою. На такій діаграмі відображаються актори (користувачі або зовнішні системи) та варіанти використання, які описують типові сценарії роботи з системою. Побудова діаграми варіантів використання дозволяє з’ясувати склад функціональних можливостей системи, чітко розмежувати ролі користувачів та визначити основні бізнес-процеси, що підлягають автоматизації.

На рисунках 2.1 – 2.4 подано діаграми варіантів використання вебзастосунку для сервісного обслуговування автомобілів. На діаграмах

виділено чотирьох основних акторів: «Клієнт», «Співробітник», «Менеджер» та «Адміністратор», які взаємодіють із системою відповідно до своїх повноважень.

Клієнт має можливість зареєструватися в системі, виконати вхід, редагувати власний профіль, створювати нові замовлення на обслуговування автомобіля, переглядати та за потреби уточнювати інформацію щодо вже створених замовлень, а також залишати запити на зворотний дзвінок. Таким чином, з точки зору клієнта система виступає як зручний інтерфейс для оформлення послуг автосервісу та відстеження стану їх виконання.

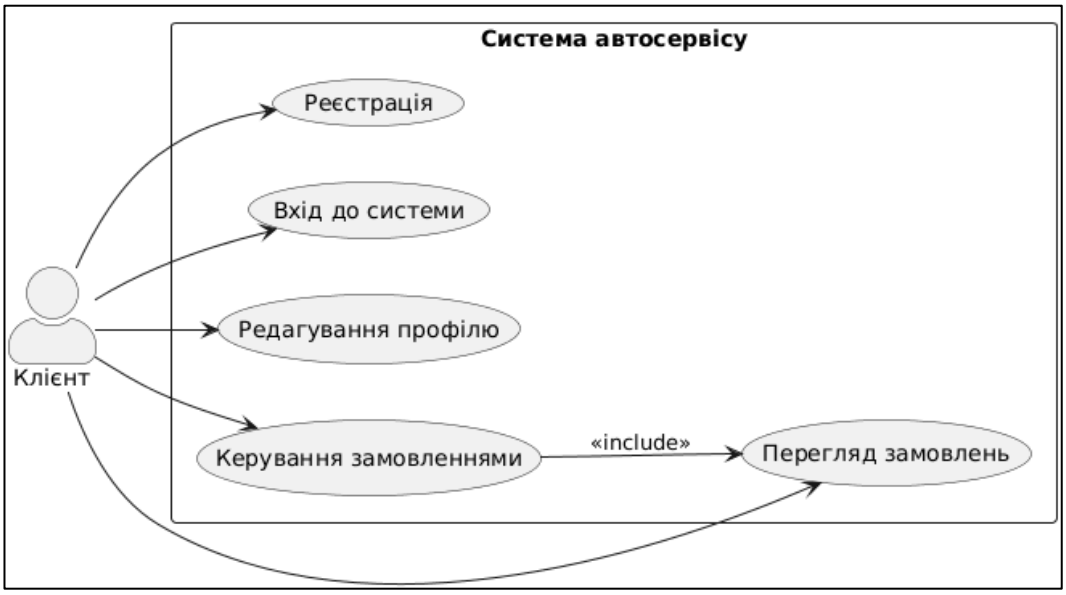


Рисунок 2.1 – Діаграма варіантів використання системи автосервісу для «Клієнта»

Співробітник автосервісу працює переважно з поточними замовленнями та завданнями (див. рис. 2.2). Він має можливість переглядати список замовлень, додавати до них конкретні завдання (роботи), фіксувати використані матеріали, змінювати статуси завдань, а також переглядати загальний перелік завдань на поточний день. Крім того, співробітник може опрацьовувати телефонні звернення клієнтів, реєструючи їх у системі та пов’язуючи із відповідними замовленнями. Таке розмежування забезпечує прозорий облік виконуваних робіт і матеріалів.

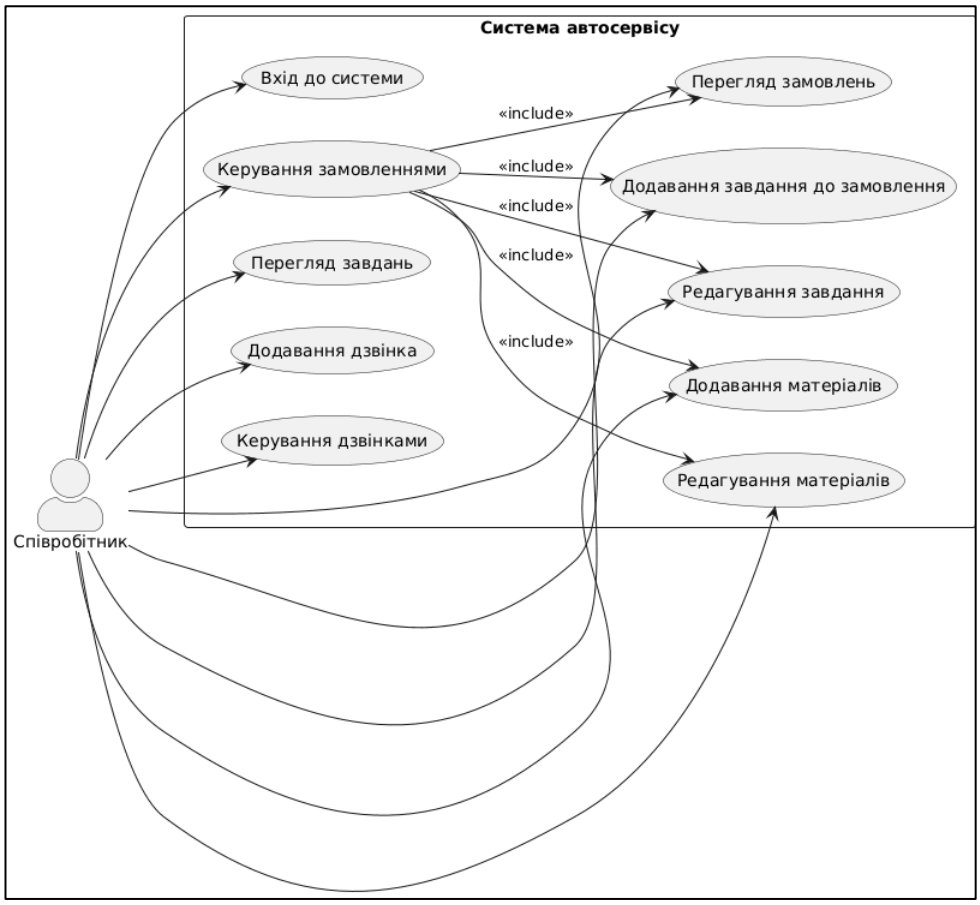


Рисунок 2.2 – Діаграма варіантів використання системи автосервісу для «Співробітника»

Менеджер має розширені повноваження щодо керування замовленнями та клієнтською базою (див. рис. 2.3). До його варіантів використання належать перегляд і редагування інформації про замовлення, видалення некоректних або помилково створених замовлень, формування рахунків за виконані роботи, перегляд сформованих рахунків, а також редагування даних клієнтів. Менеджер відповідає за коректність фінансової інформації та загальний контроль за процесом обслуговування замовлень.

Адміністратор виконує функції системного управління (див. рис. 2.4). Він може створювати та видаляти облікові записи користувачів, призначати ролі (клієнт, співробітник, менеджер, адміністратор), змінювати параметри доступу та контролювати коректність роботи системи. Таким чином, адміністратор забезпечує безпеку та цілісність даних, а також підтримує актуальність структури користувацьких ролей.

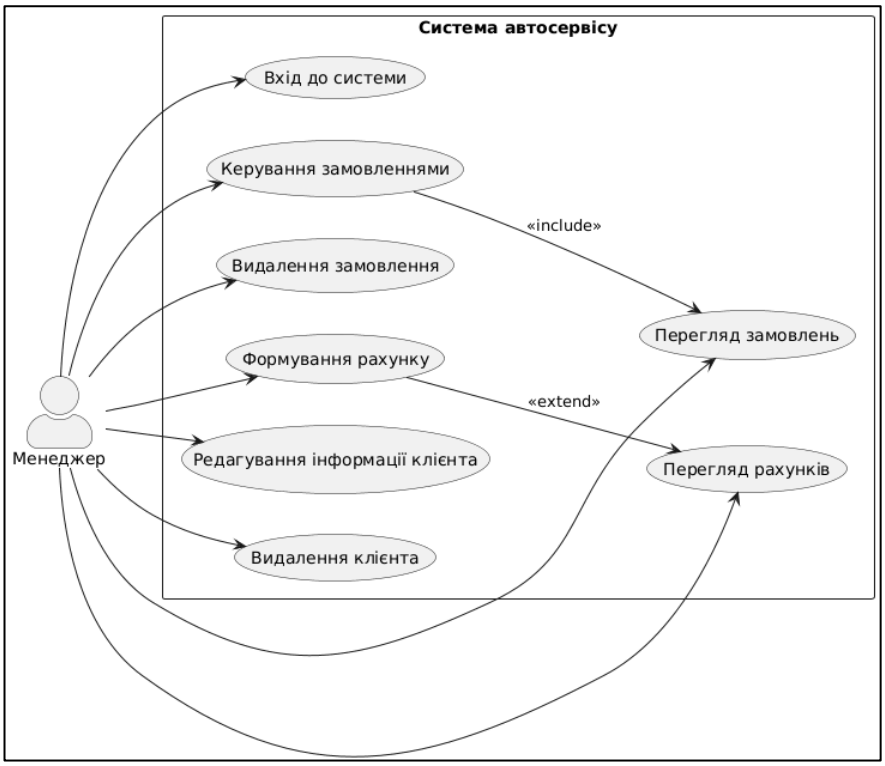


Рисунок 2.3 – Діаграма варіантів використання системи автосервісу для «Менеджера»

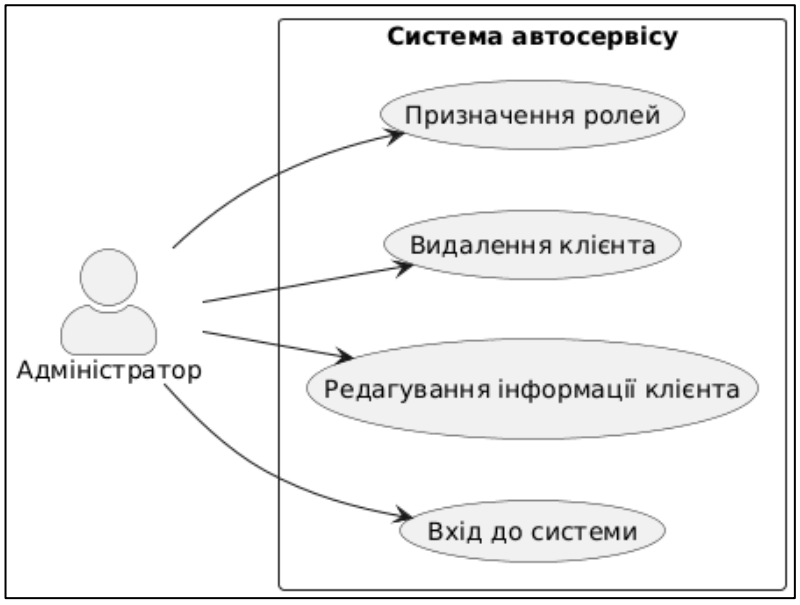


Рисунок 2.4 – Діаграма варіантів використання системи автосервісу для «Адміністратора»

На діаграмі передбачено низку базових варіантів використання, спільних для більшості акторів, зокрема: «Реєстрація», «Вхід до системи», «Редагування профілю», «Перегляд замовлень». Головним бізнес-процесом

системи є варіант використання «Керування замовленнями», який описує повний життєвий цикл замовлення: від його створення до формування рахунку та завершення робіт.

Для деталізації цього основного прецеденту використовуються відношення <<include>> та <<extend>>. Наприклад, варіанти використання «Додавання завдання до замовлення», «Додавання матеріалів», «Редагування завдання», «Редагування матеріалів», «Формування рахунку» включаються (<<include>>) до базового варіанта «Керування замовленнями», оскільки є невід'ємними складовими процесу обслуговування. Варіант «Перегляд рахунків» може розглядатися як розширення (<<extend>>) процесу формування рахунку, оскільки виконується після створення відповідних фінансових документів.

2.2.1 Опис варіантів використання

Прецедент «Керування замовленнями» *(загальний прецедент для групування дій над замовленнями)*.

Призначення: даний варіант використання надає можливість користувачам (клієнт, співробітник, менеджер) переглядати, створювати та редагувати замовлення у межах наданих роллю прав доступу.

Основний потік подій: варіант використання починається тоді, коли користувач переходить до розділу «Замовлення» з головного меню. Система відображає перелік існуючих замовлень і доступні інструменти взаємодії відповідно до ролі користувача. Користувач може переглядати деталі замовлень, обирати окреме замовлення для уточнення інформації або переходити до створення нового замовлення.

Альтернативний потік подій: якщо жодного замовлення ще не було створено, система відображає повідомлення «Немає жодного запису».

Передумова: користувач повинен виконати вхід до системи.

Прецедент «Перегляд замовлень» *(прецедент, який має зв'язок "include" з прецедентом «Керування замовленнями»)*.

Призначення: надання користувачу можливості переглядати перелік замовлень, створених ним або закріплених за ним (залежно від ролі).

Основний потік подій: прецедент активується автоматично при переході до розділу «Замовлення». Система відображає перелік замовлень, їхні статуси, дату створення та відповідальних співробітників.

Альтернативний потік подій: у разі відсутності записів система повідомляє про це відповідним текстом.

Передумова: користувач повинен бути авторизованим.

Прецедент «Додавання завдання до замовлення» *(має зв'язок "include" з батьківським прецедентом «Керування замовленнями»)*.

Призначення: надання можливості співробітнику автосервісу додавати окремі роботи (діагностика, ремонт, заміна компонентів) до вибраного замовлення.

Основний потік подій: прецедент починається після того, як співробітник відкриває конкретне замовлення та натискає кнопку «Додати завдання». Система відображає форму додавання нового завдання. Після заповнення обов'язкових полів співробітник натискає «Зберегти», після чого нове завдання прив'язується до замовлення.

Виняткова ситуація 1: у разі введення некоректних або неповних даних система повідомляє про помилку та пропонує виправити значення.

Передумова: користувач повинен мати роль співробітника чи менеджера.

Прецедент «Редагування завдання».

Призначення: надання можливості змінювати інформацію про завдання: його опис, статус або терміни виконання.

Основний потік подій: варіант використання починається при виборі співробітником конкретного завдання у межах замовлення. Система відкриває форму редагування. Після внесення змін користувач натискає «Зберегти», і система оновлює інформацію.

Виняткова ситуація: при введенні некоректних даних система відображає повідомлення про помилку.

Передумова: користувач повинен бути співробітником або менеджером.

Прецедент «Додавання матеріалів» *(має зв'язок "include" з прецедентом «Керування замовленнями»)*.

Призначення: надання можливості додавати використані витратні матеріали до замовлення.

Основний потік подій: співробітник відкриває замовлення та обирає функцію «Додати матеріали». Система відкриває форму введення даних матеріалу. Після підтвердження дані зберігаються.

Виняткова ситуація: у разі відсутності необхідних матеріалів у складі система повідомляє відповідним текстом.

Передумова: авторизація співробітника або менеджера.

Прецедент «Перегляд рахунків».

Призначення: надання можливості менеджеру переглядати перелік усіх сформованих рахунків.

Основний потік подій: після вибору відповідного розділу система завантажує перелік рахунків, відображаючи ключові параметри: номер, дату, суму, клієнта.

Передумова: користувач повинен мати роль менеджера.

Прецедент «Призначення ролей».

Призначення: надання адміністратору можливості керувати правами доступу користувачів.

Основний потік подій: адміністратор відкриває список користувачів, вибирає потрібний запис і задає роль зі списку доступних. Система зберігає зміни та оновлює статус користувача.

Передумова: авторизація адміністратора.

Прецедент «Формування рахунку» *(має зв'язок "extend" з прецедентом «Перегляд рахунків»).*

Призначення: створення рахунку за виконані роботи та використані матеріали.

Основний потік подій: менеджер обирає завершене замовлення та натискає кнопку «Сформувати рахунок». Система обчислює загальну вартість робіт та матеріалів, формує рахунок і додає його до реєстру рахунків.

Передумова: замовлення повинно бути закритим (усі завдання виконані).

2.3 Діаграма класів для прецеденту «Керування замовленнями»

Основні сутності, які беруть участь у цьому процесі:

– замовлення (Order);
– завдання (Task);
– матеріал (Material);
– клієнт (Client);
– співробітник (Staff);
– рахунок (Invoice).

І зв’язки між ними:

– клієнт створює замовлення;
– замовлення містить завдання;
– завдання можуть використовувати матеріали;
– співробітник виконує завдання;
– менеджер формує рахунок на підставі замовлення.

Діаграма класів (див. рис. 2.5) відображає структуру основних сутностей, що беруть участь у процесі «Керування замовленнями».

Клас Client описує клієнта, який має можливість створювати нові замовлення.

Клас Order містить інформацію про замовлення, його статус, опис, дату створення та пов’язаний список завдань.

Клас Task представляє окрему роботу, що виконується в межах замовлення, а клас Material описує матеріали, що були використані під час виконання роботи.

Клас Staff визначає співробітника автосервісу, який виконує завдання.

Клас Invoice пов’язаний із замовленням і формується менеджером після завершення всіх завдань.

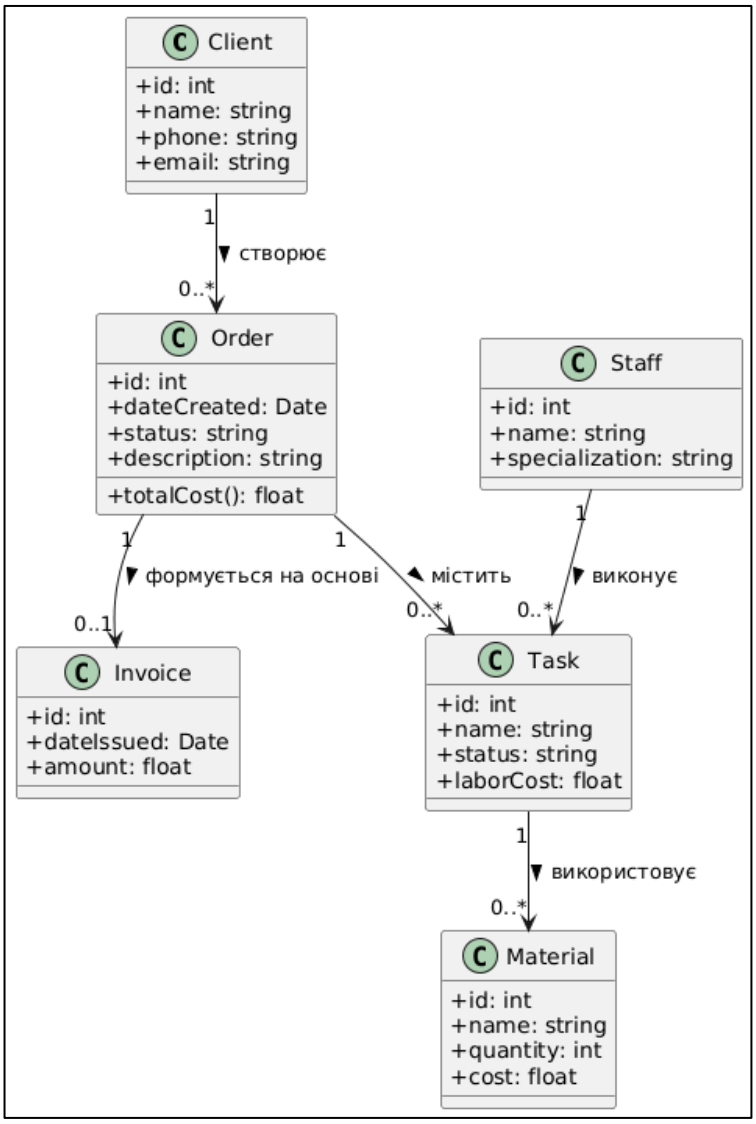


Рисунок 2.5 – Діаграма класів для прецеденту «Керування замовленнями»

2.4 Діаграма послідовності для прецеденту «Керування замовленнями»

Діаграма послідовності (див. рис. 2.6) моделює взаємодію між користувачем, вебінтерфейсом, серверною частиною та базою даних під час виконання прецеденту «Керування замовленнями».

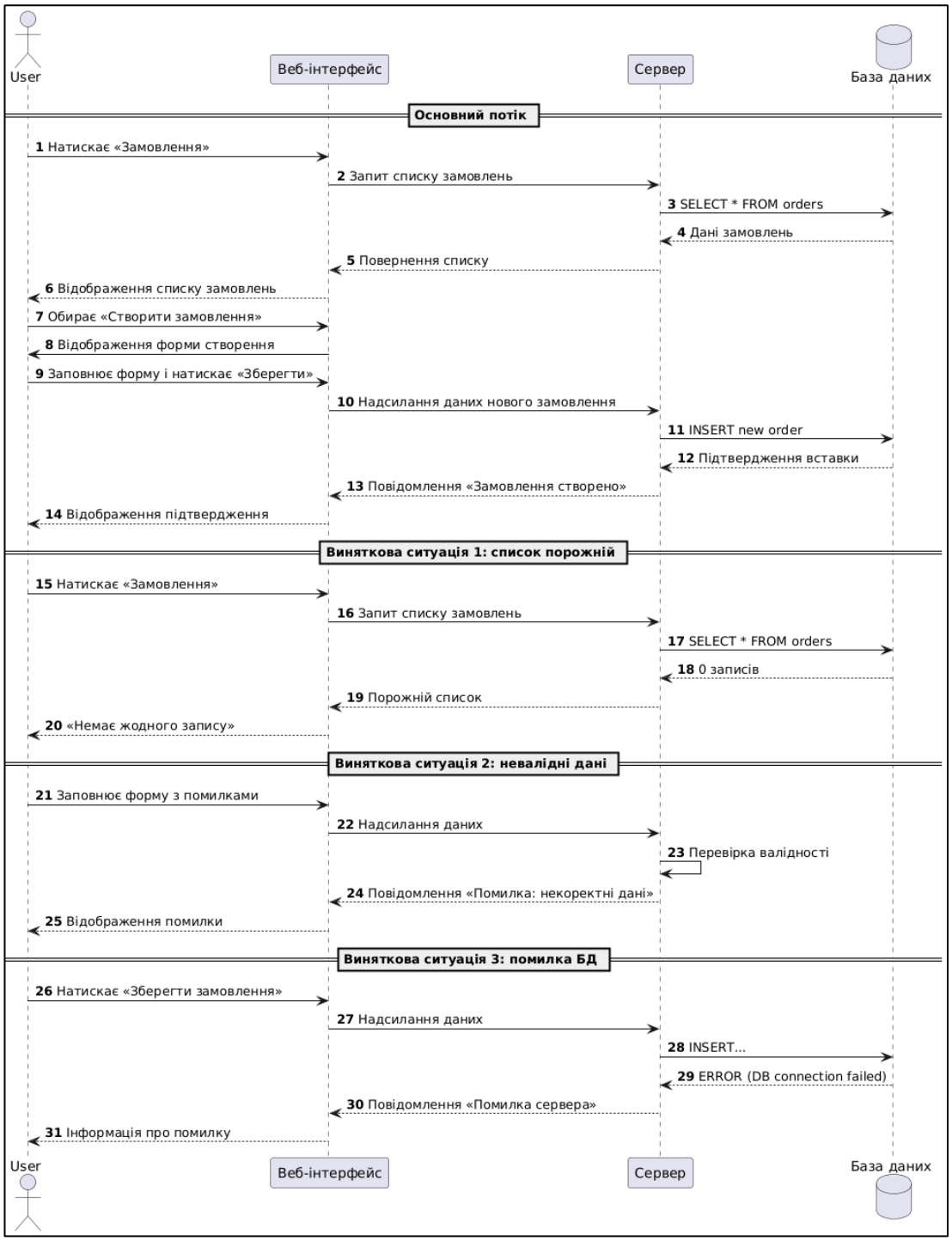


Рисунок 2.6 – Діаграма послідовності для прецеденту «Керування замовленнями»

Основний сценарій включає отримання списку замовлень, створення нового замовлення та підтвердження збереження. Також передбачено три виняткові ситуації: відсутність записів, некоректні дані користувача та помилки бази даних. Такий підхід забезпечує повне охоплення поведінки системи та дозволяє описати її реакцію на можливі відхилення.

2.5 Діаграма розгортання для системи автосервісу

Діаграма розгортання (див. рис. 2.7) відображає фізичну архітектуру вебзастосунку автосервісу, розробленого за технологічною моделлю MERN (MongoDB, Express.js, React, Node.js). У системі виділено чотири основні рівні: клієнтський рівень, рівень фронтенду, серверний рівень і рівень бази даних.

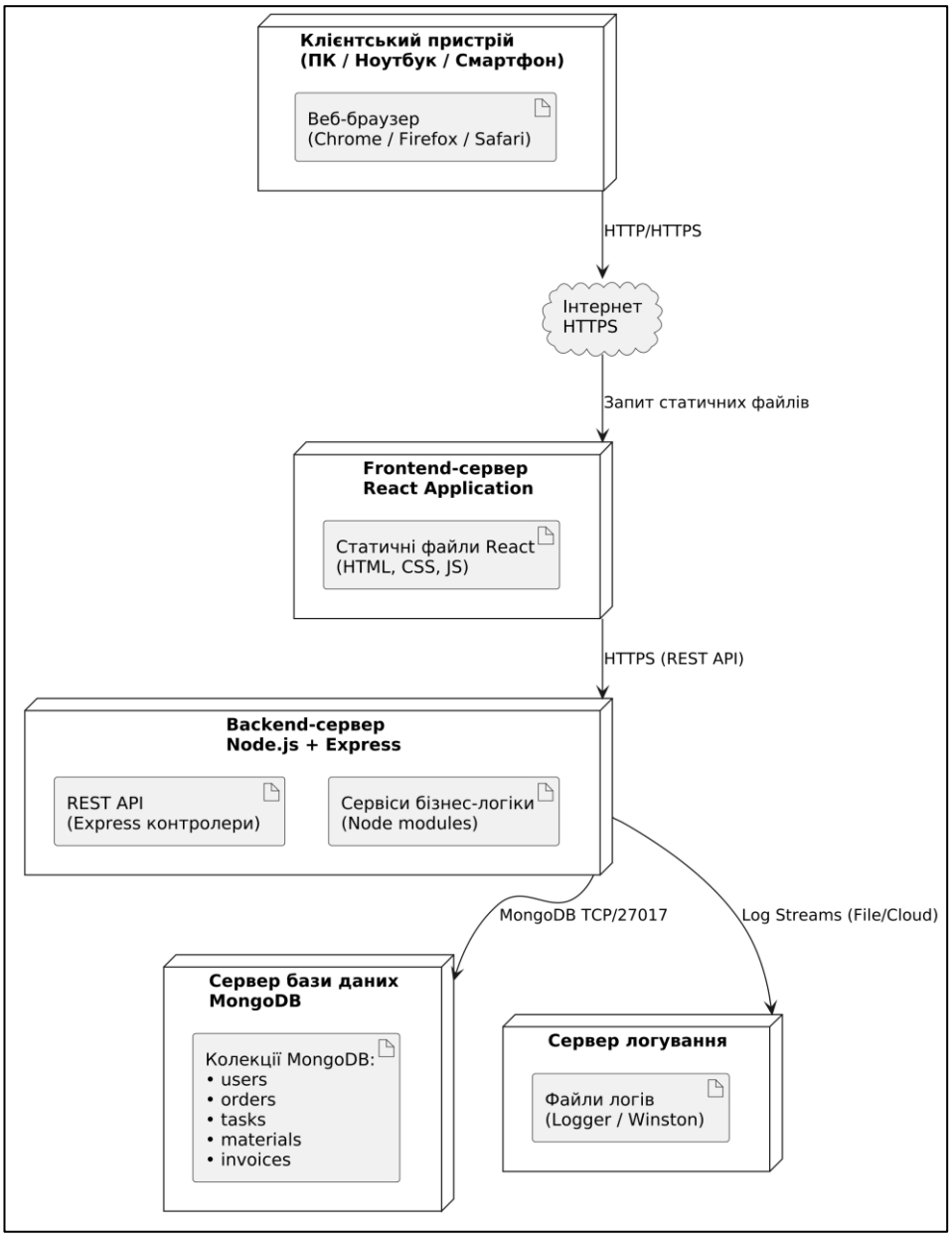


Рисунок 2.7 – Діаграма розгортання

На клієнтському пристрої виконується веббраузер, який забезпечує інтерфейс користувача та здійснює доступ до фронтендсервера через протокол

HTTPS. На фронтендсервері розгорнуто статичні файли React застосунку, які завантажуються браузером та виконуються на стороні клієнта.

Серверна логіка реалізована на базі Node.js та Express.js і працює на окремому backend сервері. На ньому розміщені модулі бізнеслогіки та REST контролери, які обробляють запити клієнтського застосунку. Для зберігання даних використовується окремий сервер бази даних MongoDB, що містить відповідні колекції (users, orders, tasks, materials, invoices).

Для забезпечення надійності та можливості аудиту окремо функціонує сервер логування, на якому зберігаються журнали роботи системи (Winston або інші логгери). Усі елементи взаємодіють через захищені мережеві з’єднання відповідно до сучасних вимог інформаційної безпеки.

Застосування централізованого логування дозволяє відстежувати дії користувачів та системних компонентів, оперативно виявляти помилки, здійснювати аналіз продуктивності й забезпечувати відновлення працездатності у разі виникнення збоїв.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Технологічний стек MERN (MongoDB, Express.js, React, Node.js) є одним із найпоширеніших підходів до розробки сучасних вебзастосунків. Його популярність обумовлена високою продуктивністю, гнучкістю масштабування та можливістю створювати повноцінні SPA-додатки на JavaScript [5, 6].

Ефективність MERN-архітектури у реальних прикладних проєктах підтверджується низкою кваліфікаційних робіт. Наприклад, у роботі Марчука Д. В. реалізовано платформу для блогів з використанням React, Node.js та MongoDB [7]. У дослідженні Шальнєва О. В. показано, що MERN дозволяє створювати високопродуктивні торгові вебзастосунки з адаптивними інтерфейсами та зручним керуванням даними [8].

У професійній літературі MERN-стек часто порівнюється з іншими стеками, такими як MEAN або класичний Full-Stack (LAMP). Згідно з дослідженням Продоляка Б. [9], MERN демонструє кращу продуктивність у SPA-додатках завдяки повністю JavaScript-орієнтованій архітектурі. У матеріалах EasyShiksha підкреслюється, що MERN надає найшвидший шлях до створення модульних повнофункціональних застосунків [10].

Для керування глобальним станом застосунку використано Redux Toolkit, а синхронізацію сторінок і модулів навігації реалізовано за допомогою React Router DOM. Взаємодія з сервером здійснюється за допомогою бібліотеки Axios. Для генерації унікальних ідентифікаторів використано пакет uuid та uuidv4. Тестування інтерфейсу виконувалося через бібліотеки @testing-library/react та @testing-library/jest-dom.

Фронтенд-застосунок зібрано з використанням React [12]. Офіційні рекомендації щодо синтаксису, роботи з промісами, об’єктами та подіями

наведені у документації Mozilla Developer Network [11], яка є одним із найавторитетніших джерел у сфері веброзробки.

Для забезпечення сучасного візуального оформлення застосовано бібліотеку MUI (Material UI) зі стильовою системою Emotion (@emotion/react, @emotion/styled) та іконковим набором @mui/icons-material. Типізація фронтенд-частини реалізована на основі TypeScript.

Серверну частину застосунку реалізовано на основі Node.js з використанням фреймворку Express.js, дозволяє ефективно реалізовувати REST API та обробку асинхронних операцій [13].обробки HTTP-запитів та побудови middleware. Для обробки даних запитів використано body-parser, а для забезпечення взаємодії між клієнтом і сервером CORS.

Зберігання даних здійснюється в документно-орієнтованій базі даних MongoDB, а роботу зі схемами та моделями реалізовано за допомогою ORM-бібліотеки Mongoose. Для хешування паролів у процесі автентифікації використано бібліотеку bcrypt, а для токен-орієнтованої авторизації стандарт JSON Web Token (JWT). Для обробки завантаження файлів застосовано middleware-бібліотеку Multer.

3.2 Основні компоненти системи

Оскільки основним прецедентом системи є «Керування замовленнями», основні компоненти системи визначаються тими компонентами, які забезпечують створення, редагування, видалення та перегляд інформації про замовлення, завдання, матеріали, клієнтів та рахунки.

Архітектура MERN передбачає розподіл відповідальності між фронтенд- та бекенд-частиною, тому ключові компоненти розглядаються окремо.

3.2.1 Основні функціональні модулі фронтенд-частини

Фронтенд-частина системи реалізована на основі React, Redux Toolkit та TypeScript. Оскільки основним прецедентом є «Керування замовленнями», ключові функціональні модулі відповідають за створення, редагування, перегляд і видалення замовлень та пов’язаних із ними завдань.

Основними елементами фронтенд-архітектури є:

– Redux-модулі (ordersSlice, tasksSlice), реалізують бізнес-логіку, зберігають стан застосунку та виконують асинхронні CRUD-операції;
– React-компоненти (OrdersList, OrderFormAdd, OrderCard), відповідають за графічне відображення інформації та взаємодію користувача із системою.

Призначення модулю ordersSlice:

– зберігає список замовлень у Redux Store;
– виконує CRUD-операції через асинхронні Thunk-функції;

Основний потік подій:

– компонент викликає fetchOrders() (див. рис. 3.1);
– модуль надсилає HTTP-запит до API;
– отримані дані записуються у стан (setOrders);
– компоненти отримують оновлений стан (наприклад OrdersList).

```
export const fetchOrders = () => async (dispatch) => {
const response = await instance.get('/orders');
dispatch(setOrders(response.data));
```

Рисунок 3.1 – Приклад фрагмента модуля ordersSlice

Структурні елементи модуля:

– інтерфейс Order: визначає структуру об’єкта замовлення;
– reducers: addOrder, updateOrder, deleteOrder, setOrders;
– thunks: fetchOrders, createOrder, modifyOrder, removeOrder.

Призначення модулю tasksSlice:

– керує завданнями, що прив’язані до конкретного замовлення;
– виконує операції створення, редагування статусу, зміни матеріалів та видалення;
– забезпечує синхронізацію з бекендом.

Основний потік подій:

– при створенні замовлення може бути створене завдання (addTaskSank);
– при видаленні замовлення видаляються всі відповідні завдання (removeOrderTasksSank);
– зміна статусу або тексту завдання оновлює Redux і бекенд.

Структурні елементи:

– тип TaskType – опис структури завдання;
– reducers – addTask, removeTask, removeAllOrderTasks, editTaskStatus, editTaskTitle, editTaskSupplies;
– thunks – initTasks, addTaskSank, removeTaskSank, updateTaskStatusSank (див. рис. 3.2).

```
export const updateTaskStatusSank = (id, isDone) => async (dispatch) => {
await instance.put(`/tasks/changeTaskStatus/${id}`, {isDone});
dispatch(editTaskStatus({id, isDone}));
};
```

Рисунок 3.2 – Приклад фрагмента модуля tasksSlice

Призначення компоненту OrdersList:

– відображає список замовлень та ініціалізує їх завантаження (див. рис. 3.3);
– забезпечує інтеграцію з Redux-сховищем.

Основний потік роботи:

– після автентифікації викликається fetchOrders() для оновлення даних;
– компонент рендерить список карток замовлень;

– на сторінці доступна форма для створення нового замовлення (OrderFormAdd);
– якщо користувач не авторизований – показується сторінка About.

```
useEffect(() => {
if (isAuthenticated) dispatch(fetchOrders());
}, [isAuthenticated]);
```

Рисунок 3.3 – Приклад використання useEffect

Призначення компоненту OrderFormAdd: надає користувачу інтерфейс для створення нового замовлення.

Основний потік подій:

– користувач заповнює поля форми;
– формується об’єкт Order;
– викликається createOrder() з модуля ordersSlice;
– список замовлень оновлюється.

Призначення компоненту OrderCard:

– відображає окреме замовлення у вигляді картки;
– надає інструменти для редагування та видалення.

Основні функції:

– візуалізація інформації про замовлення;
– виклик modifyOrder() при редагуванні;
– виклик removeOrder() при видаленні.

Нижче наведено UML-діаграму модулів й компонентів (див. рис. 3.4), яка відображає взаємодію між основними структурами даних, Redux-слайсами та React-компонентами.

Діаграма дозволяє візуально оцінити поток взаємодій між React-компонентами, Redux-слайсами та структурою даних, що лежить в основі функціонування системи автосервісу.

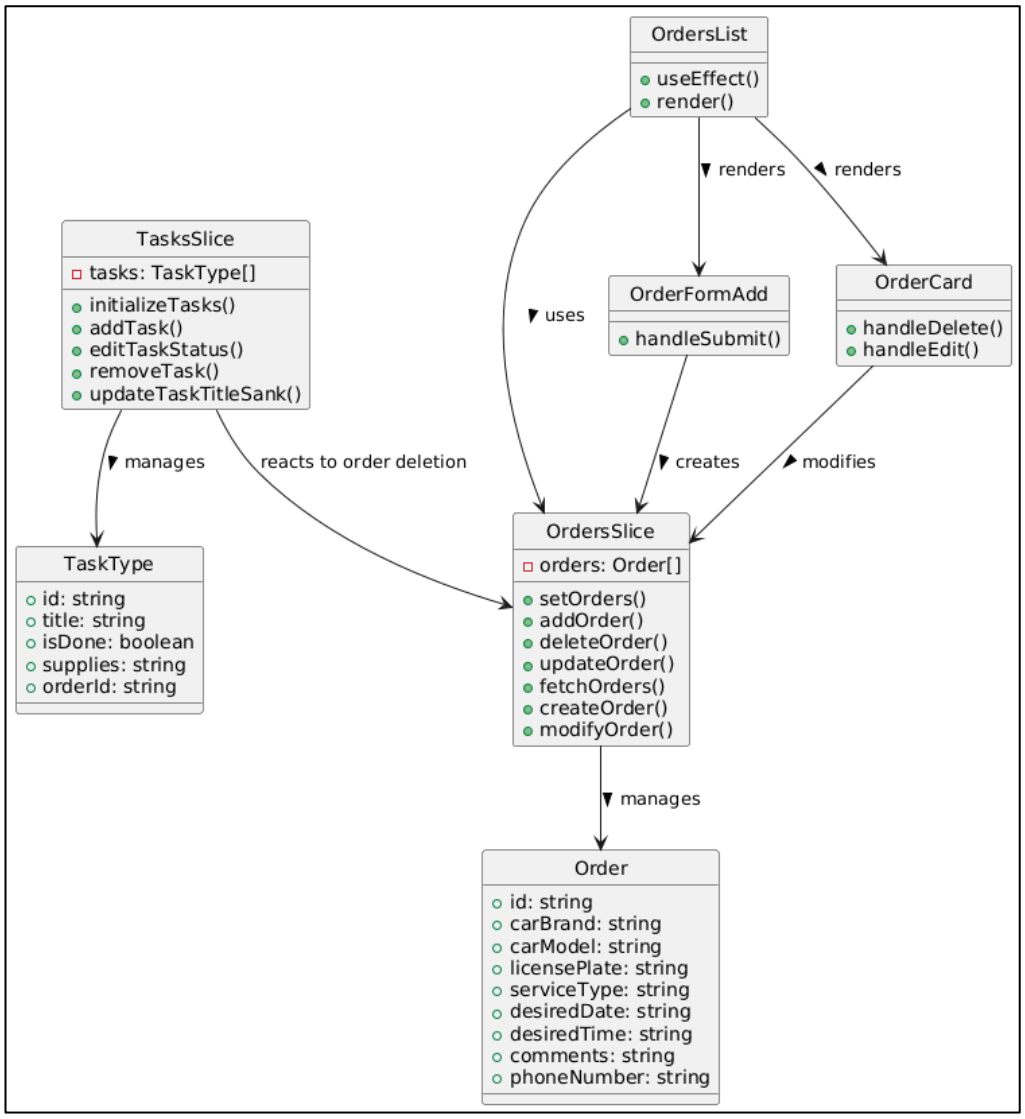


Рисунок 3.4 – Діаграма основних модулів й компонентів фронтенда

3.2.2 Основні функціональні модулі бекенд-частини

Для побудови REST API застосовано Node.js та Express. Офіційна документація Node.js [13] описує неблокуючу модель введення-виведення, яка забезпечує високу продуктивність серверної частини. MongoDB Manual [14] використано для розробки схеми зберігання даних про замовлення, клієнтів і інвойси.Відповідно до головного прецеденту «Керування замовленнями» є:

– Express-маршрути (ordersRoutes, tasksRoutes), приймають HTTP-запити та викликають відповідні контролери;

– контролери (orderController, tasksController), реалізують бізнес-логіку: створення, редагування, видалення та отримання даних;
– Mongoose-моделі (Order, Task), визначають структуру даних MongoDB;
– проміжні middleware-модулі (verifyToken, checkRole, checkOrderAccess), забезпечують автентифікацію та авторизацію.

Призначення модулю Order (Mongoose-модель):

– визначає структуру документа «замовлення» у MongoDB;
– використовується контролером для створення, оновлення та пошуку замовлень;
– забезпечує валідацію даних відповідно до схеми.

Основні поля моделі (див. рис. 3.5):

– userId (ідентифікатор користувача);
– id (унікальний UUID замовлення);
– carBrand, carModel, licensePlate (дані про автомобіль);
– serviceType (обраний тип послуги);
– desiredDate, desiredTime (бажані дата та час);
– comments (додаткова інформація клієнта);
– phoneNumber (контактний номер).

```
const orderSchema = new mongoose.Schema({
userId: {type: String, required: true},
id: {type: String, required: true, unique: true},
carBrand: {type: String, required: true},
carModel: {type: String, required: true},
licensePlate: {type: String, required: true},
serviceType: {type: String, required: true},...
```

Рисунок 3.5 – Фрагмент схеми orderSchema

Призначення модулю Task (Mongoose-модель (див. рис. 3.6)):

– зберігає інформацію про завдання, що належать до конкретного замовлення;

– використовується для управління етапами виконання робіт.

Основні поля:

– id (унікальний ідентифікатор завдання);

– orderId (зв’язок із замовленням);

– title (назва роботи);

– supplies (матеріали);

– isDone (статус виконання).

```
const taskSchema = new mongoose.Schema({
  id: {type: String, required: true},
  orderId: {type: String, required: true},
  title: {type: String, required: true},
  supplies: {type: String, default: "Add supplies"},
  isDone: {type: Boolean, default: false},
});
```

Рисунок 3.6 – Фрагмент схеми taskSchema

Призначення контролеру orderController:

– реалізує логіку отримання (див. рис. 3.7), створення, редагування та видалення замовлень;

– враховує роль користувача (клієнт, співробітник, менеджер);

– перевіряє права доступу до конкретного замовлення.

```
exports.getOrders = async (req, res) => {
  const userRole = req.userRole;
  if (userRole === 'manager') {
    const orders = await Order.find();
    return res.json(orders);
  }
  const orders = await Order.find({ userId: req.user.id });
  res.json(orders);
};
```

Рисунок 3.7 – Фрагмент логіки для отримання замовлень

Основний потік подій:

– користувач надсилає HTTP-запит до API (/orders);
– Middleware verifyToken розшифровує токен користувача;
– Middleware checkRole визначає права користувача;
– контролер виконує дію (пошук, створення або редагування);
– результат повертається у JSON-форматі.

Призначення контролеру tasksController:

– керує завданнями, що пов’язані із замовленнями;
– дозволяє додавати завдання, змінювати назву, матеріали та статус;
– дозволяє видаляти завдання, а також масово видаляти всі завдання замовлення.

Основні функції:

– addTask;
– deleteTask;
– deleteTasks (видалити всі завдання замовлення);
– updateTaskStatus (див. рис. 3.8);
– updateTaskTitle;
– updateTaskSupplies.

```
const updatedTask = await Task.findOneAndUpdate(
   { id: req.params.id },
   { isDone: req.body.isDone },
   { new: true }
);
```

Рисунок 3.8 – Фрагмент оновлення статусу замовлення

Призначення Express-маршрутів (orderRoutes (див. рис. 3.9), tasksRoutes (див. рис. 3.10)):

– маршрутизують HTTP-запити до відповідних контролерів;
– реалізують автентифікацію та авторизацію через middleware.

```
router.get('/orders', verifyToken, checkRole, getOrders);
router.post('/orders', verifyToken, addOrder);
router.put('/orders/:id', verifyToken, checkRole, checkOrderAccess, updateOrder);
router.delete('/orders/:id', verifyToken, checkRole, checkOrderAccess, deleteOrder);
```

Рисунок 3.9 – Фрагмент маршруту замовлень

```
router.post('/tasks', verifyToken, checkRole, checkAccessManOrStaff, addTask);
router.delete('/tasks/:id', verifyToken, checkRole, checkAccessManOrStaff, deleteTask);
```

Рисунок 3.10 – Фрагмент маршруту завдань

Діаграма основних бекенд-функціональних модулів представлена на рисунку 3.11.

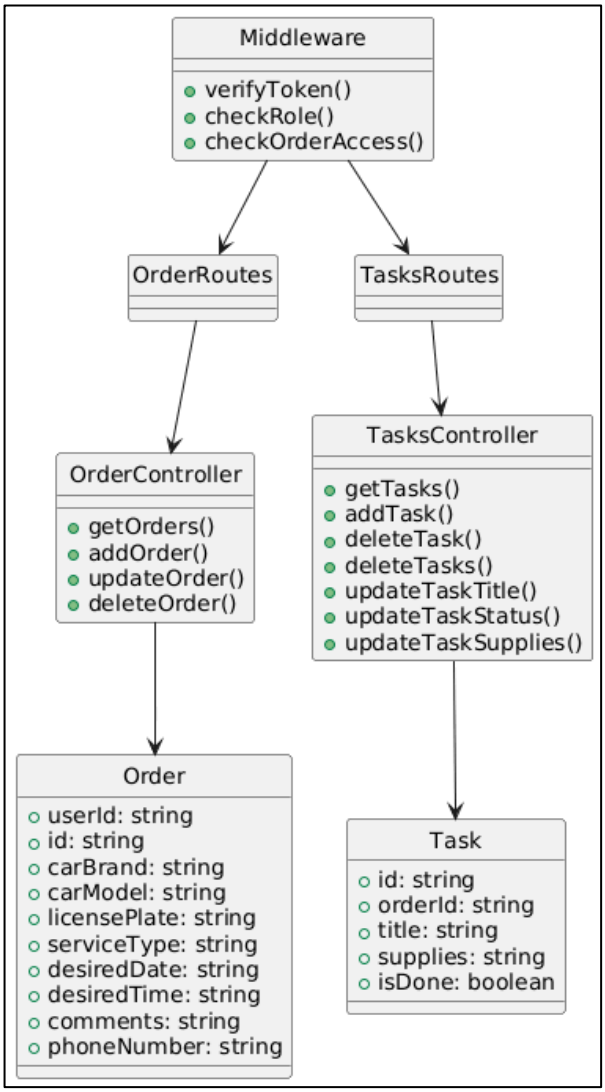


Рисунок 3.11

3.2.3 Схема роботи бази даних

База даних вебзастосунку для сервісного обслуговування автомобілів реалізована на основі документоорієнтованої системи керування базами даних MongoDB, що забезпечує гнучку структуру зберігання даних, горизонтальну масштабованість та високу продуктивність при роботі з великими обсягами напівструктурованої інформації. MongoDB Manual [14] використано для розробки схеми зберігання даних про користувачів, замовлення, завдання, рахунки, статті та телефонні звернення. На відміну від класичних реляційних систем, де структура таблиць є жорстко фіксованою, документоорієнтована модель MongoDB дозволяє зберігати гнучкі JSON-подібні документи, що ідеально підходить для роботи з об’єктами, структура яких може змінюватися залежно від бізнес-процесів автосервісу. Наприклад, різні замовлення можуть містити відмінний набір робіт, різну кількість витратних матеріалів або додаткові параметри, що не потребують зміни загальної схеми бази даних. Кожна сутність представлена у вигляді колекції, записи якої мають документну структуру з унікальними ідентифікаторами. Надалі подано опис кожної колекції та логічних зв’язків між ними.

Колекція Users призначена для зберігання даних про користувачів інформаційної системи, включаючи клієнтів, співробітників, менеджерів та адміністратора. Вона містить такі основні групи атрибутів:

– ідентифікаційні дані: id, mail, phone;
– персональні дані: firstName, lastName, patronymic, photo;
– службові дані: role, registrationDate, password.

Поле role реалізує механізм розмежування доступу, що визначає можливості користувача в межах системи. Користувач може мати багато замовлень та багато сформованих рахунків.

Колекція Orders містить інформацію про замовлення на ремонт або обслуговування автомобілів. Кожен документ містить такі атрибути:

– дані замовника: userId, phoneNumber;
– дані автомобіля: carBrand, carModel, licensePlate;

– опис послуги: serviceType, comments;

– часові параметри: desiredDate, desiredTime;

– службові дані: id, timestamps.

Поле userId встановлює зв’язок замовлення з користувачем. Кожне замовлення може містити певну кількість робіт (Tasks) та може мати пов’язаний рахунок (Invoice).

Колекція Tasks представляє перелік робіт, що виконуються у межах конкретного замовлення. До основних атрибутів належать:

– ідентифікатор завдання: id;

– ідентифікатор замовлення: orderId;

– опис роботи: title;

– матеріали: supplies, suppliesCost;

– стан виконання: isDone.

Поле orderId забезпечує зв’язок завдання з відповідним замовленням.

Колекція Invoices містить інформацію про фінальні рахунки, що формуються на основі замовлень. Структура документа включає:

– реквізити клієнта: firstName, lastName, patronymic, phoneNumber;

– реквізити автомобіля: carBrand, carModel, licensePlate;

– дані замовлення: orderId, serviceType, comments;

– перелік виконаних робіт та витратних матеріалів: tasks[];

– інформацію про вартість: workCost, totalCost, paymentDone.

Особливістю цієї колекції є те, що вона містить копію переліку завдань із замовлення на момент створення рахунка. Це відповідає вимогам щодо формування фінансових документів як фіналізованих записів.

Колекція Articles використовується для зберігання інформаційних матеріалів та новин автосервісу. Вона містить такі атрибути:

– заголовок: title;

– короткий опис: preview;

– повний текст: content;

– зображення: imageUrl, fullImageUrl;

– джерело: source, sourceName;

– службовий ідентифікатор: id.

Колекція є автономною і не має зв’язків з іншими сутностями системи.

Колекція Phones використовується для зберігання записів вхідних телефонних звернень клієнтів. Основні поля:

– ідентифікатор: id;

– номер телефону: number;

– дата звернення: date;

– ознака опрацьованого звернення: called.

Колекція забезпечує реєстрацію та контроль опрацювання телефонних заявок менеджером або співробітником.

Структура зв’язків бази даних. Логічні зв’язки між основними сутностями реалізовано засобами зберігання зовнішніх ідентифікаторів:

– User – Order: зв’язок 1→N, користувач може мати багато замовлень;

– Order – Task: зв’язок 1→N, кожне замовлення складається з набору робіт;

– Order – Invoice: зв’язок 1→1, для кожного замовлення генерується один рахунок;

– Invoice – Task: композиція, копія завдань включається до фінального документа;

– User – Invoice: зв’язок 1→N, користувач може отримати декілька рахунків.

Використання зовнішніх ключів та композиційних зв’язків дозволяє гарантувати, що кожна операція – від реєстрації замовлення до формування рахунку – виконується послідовно та з дотриманням логічної цілісності даних. У результаті модель даних підтримує прозорий, формалізований рух інформації між користувачами, замовленнями, виконаними роботами та фінальними документами автосервісу.

Діаграма на рисунку 3.12 відображає повну структуру цих взаємозв’язків.

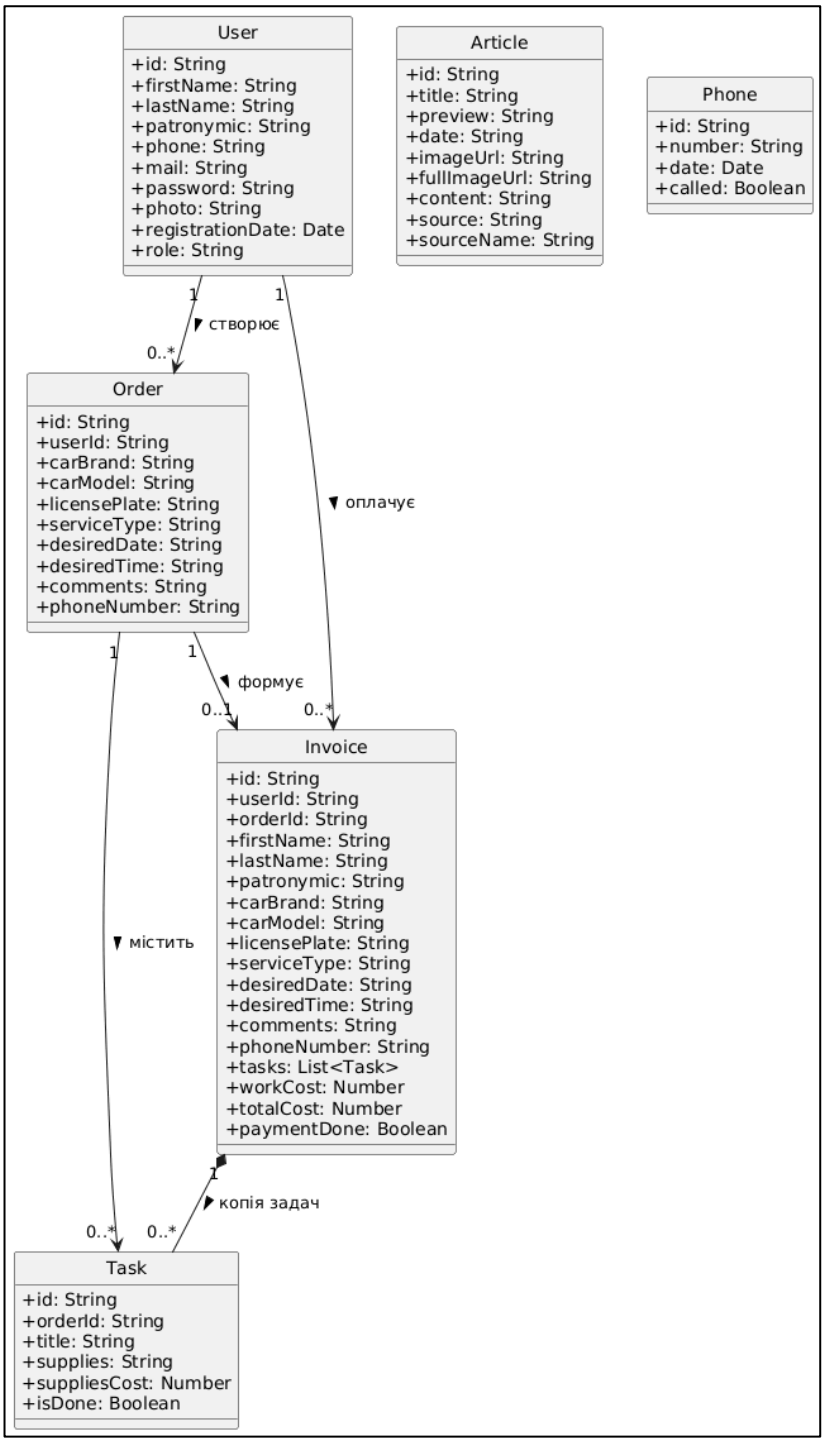


Рисунок 3.12 – Діаграма структури бази даних (MongoDB)

3.3 Тестування проєкту

У межах системного тестування було створено комплекс тестових сценаріїв, що охоплюють критично важливі бізнес-процеси вебзастосунку «AVTOREPAIR». Нижче наведено ключові з них.

Сценарій 1. Реєстрація нового користувача. Мета тесту: перевірити можливість створення нового облікового запису клієнта з унікальною електронною поштою.

Вхідні дані:

– ім’я: Олександр;

– телефон: 0991234567;

– e-mail: client@example.com;

– пароль: Test12345.

Очікуваний результат: обліковий запис створено, користувача перенаправлено на головну сторінку з автоматичним входом у систему.

Фактичний результат: система приймає дані (див. рис. 3.13), успішно реєструє користувача (див. рис. 3.14) та додає запис до колекції Users у MongoDB (див. рис. 3.15). Після реєстрації користувача автоматично авторизовано (JWT-токен збережено у LocalStorage).

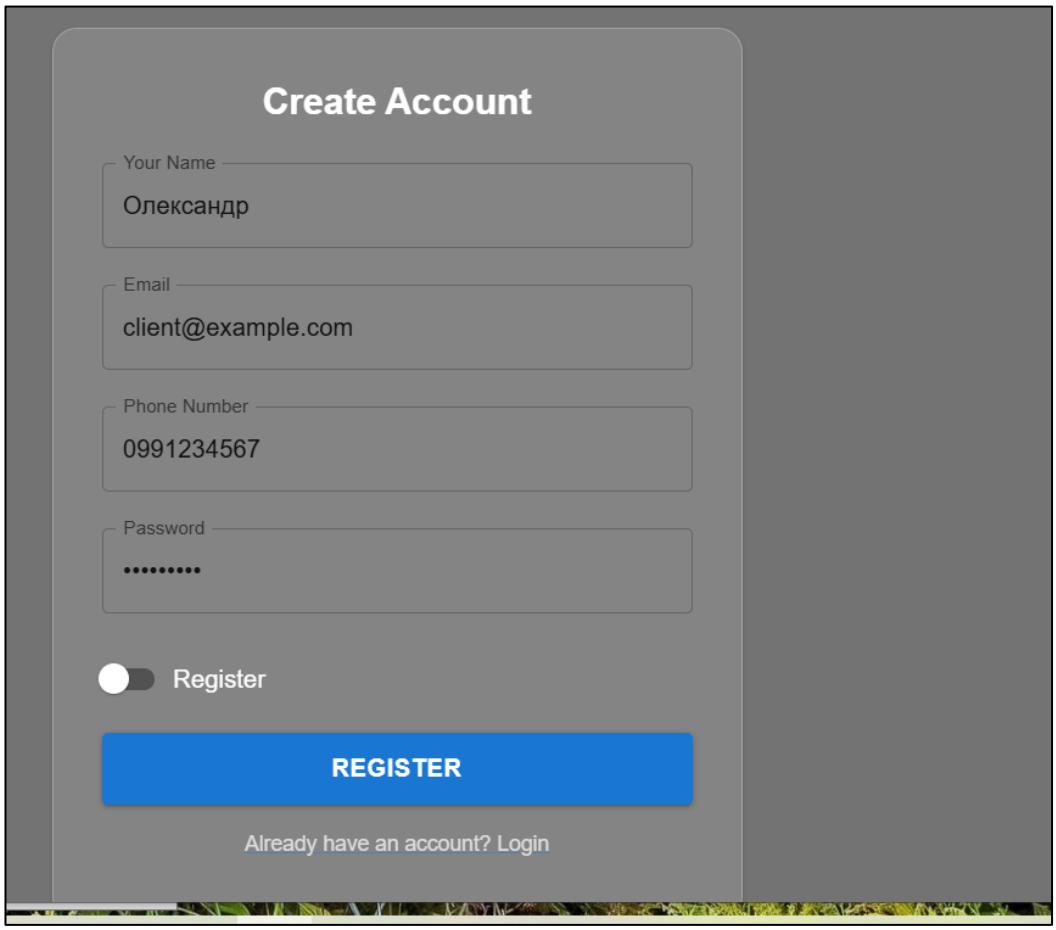


Рисунок 3.13 – Форма реєстрації нового користувача

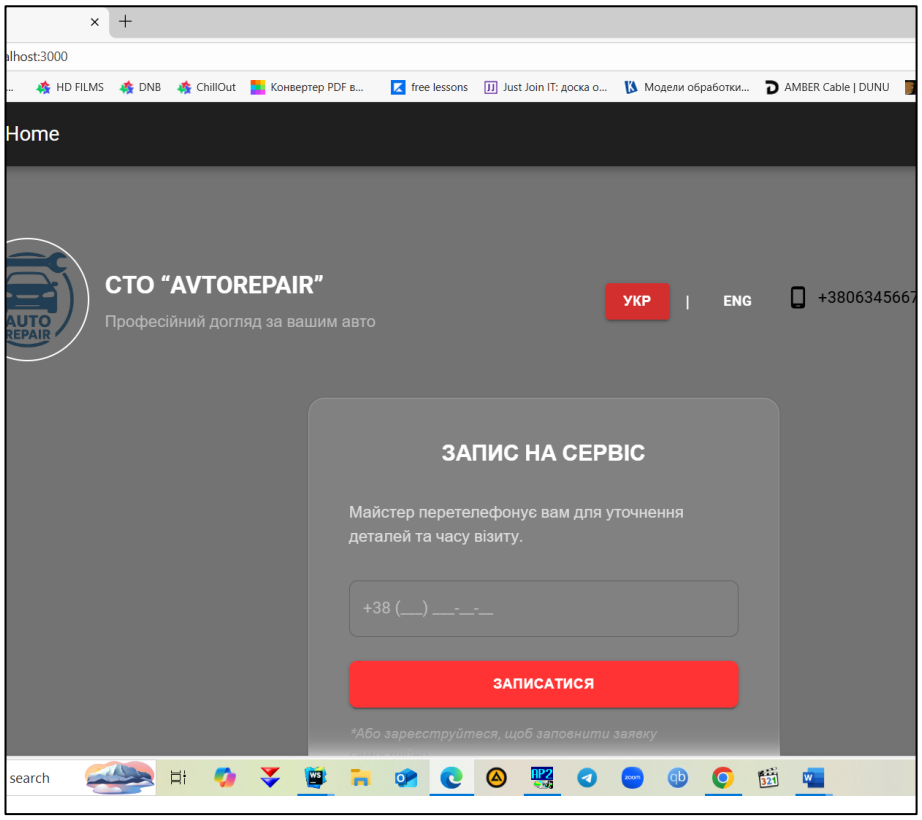


Рисунок 3.14 – Успішне надсилання форми та перенаправлення після реєстрації


```
{
  "_id": {
    "$oid": "692db33293d1d644fd4c4427"
  },
  "firstName": "Олександр",
  "lastName": "",
  "patronymic": "",
  "phone": "0991234567",
  "photo": "",
  "id": "f4871a57-024e-414b-b7a6-7334bf61abc3",
  "mail": "client@example.com",
  "password":
"$2b$10$yfpX81PPgpSSUECzTvtY4OpPl4/qBUynWL0tpz4Cj4KJ8yueO.C0C",
  "registrationDate": {
    "$date": "2025-12-01T15:24:34.128Z"
  },
  "role": "customer",
  "__v": 0
}
```


Рисунок 3.15 – Новий запис користувача у колекції Users (MongoDB Compass)

Сценарій 2. Авторизація зареєстрованого користувача. Мета тесту: перевірити можливість входу в систему з коректними обліковими даними.

Вхідні дані (див. рис. 3.16):

– e-mail: client@example.com;

– пароль: Test12345.

Очікуваний результат: користувач отримує доступ до сторінки з переліком замовлень.

Фактичний результат: система перевіряє введені дані, генерує JWT-токен та перенаправляє користувача (див. рис. 3.17).

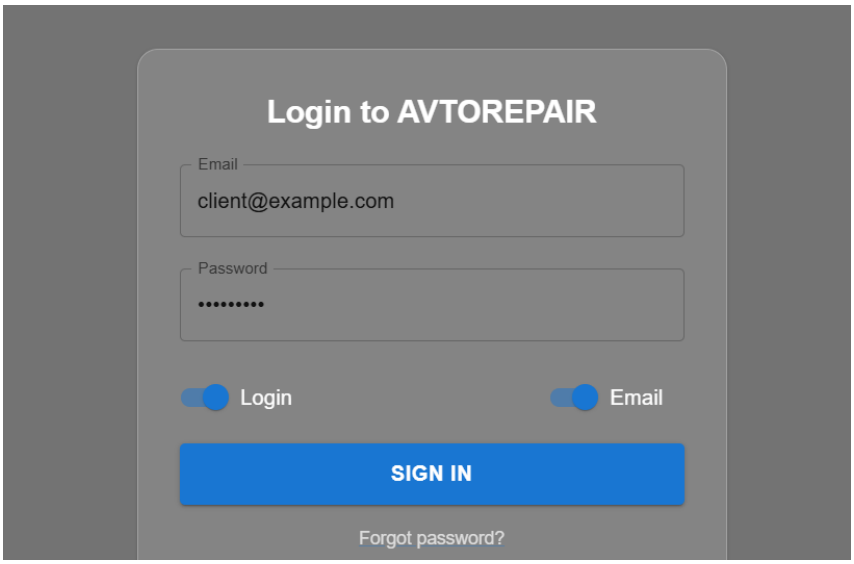


Рисунок 3.16 – Форма входу в систему

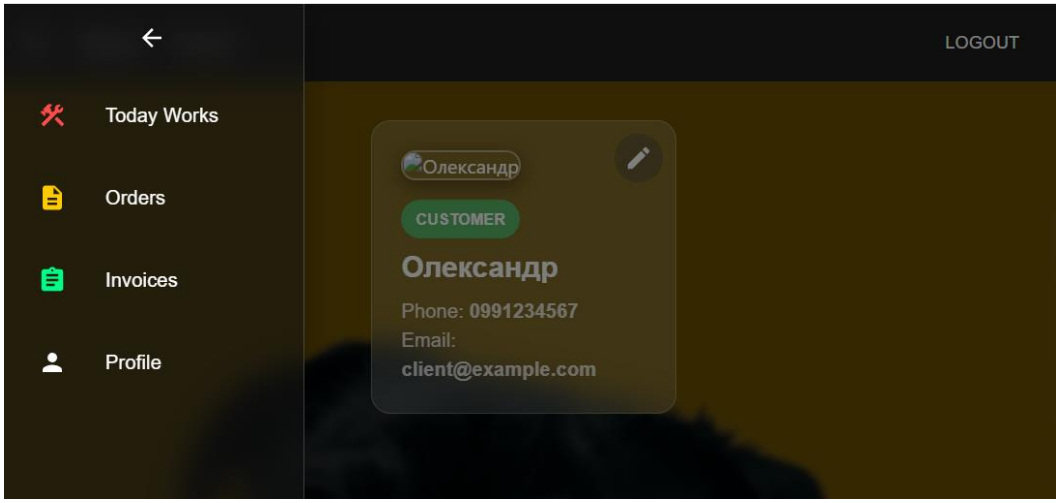


Рисунок 3.17 – Успішна авторизація користувача, перенаправлення та робочий інтерфейс після входу в систему

Сценарій 3. Створення нового замовлення. Мета тесту: перевірити коректність створення нового замовлення та синхронізацію фронтенд/бекенд.

Вхідні дані (див. рис. 3.18):

– марка авто: Toyota;

– модель: Corolla;

– номер: AA1234BK;

– тип послуги: Діагностика;

– бажана дата: 2025-12-15;

– час: 10:30;

– телефон: 0991234567.

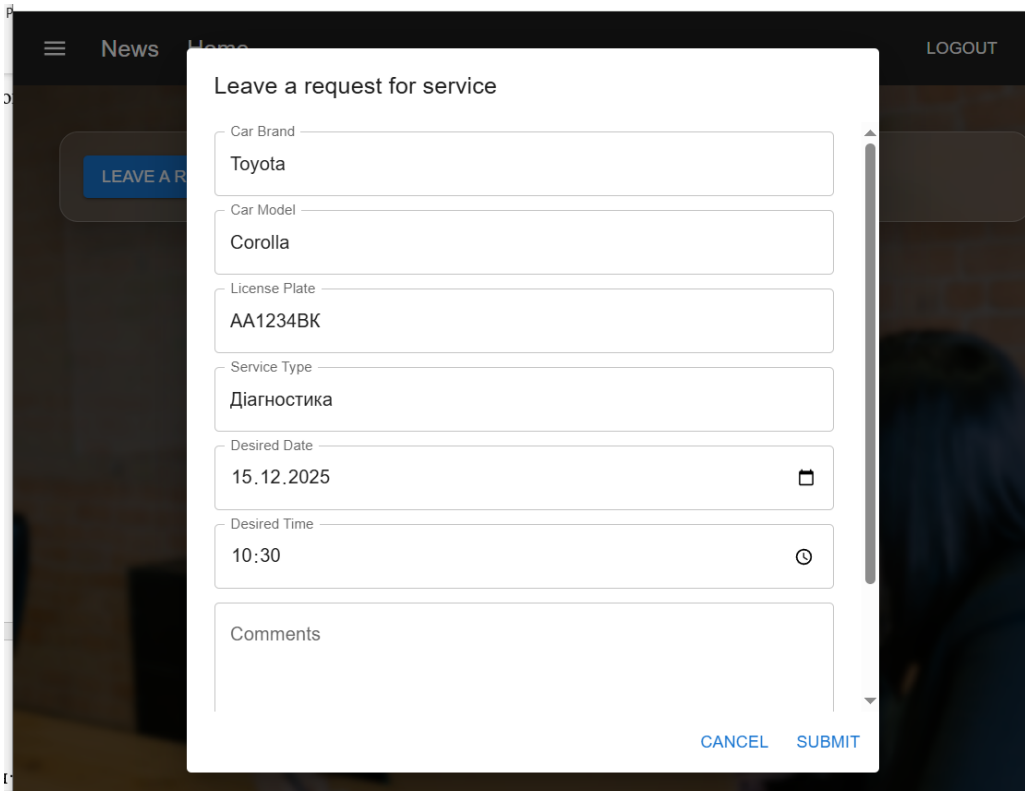


Рисунок 3.18 – Форма створення нового замовлення

Очікуваний результат: замовлення додається до бази даних та відображається у списку на фронтенді (див. рис. 3.19).

Фактичний результат: функція createOrder() викликає POST-запит /orders, бекенд формує документ у колекції Orders, слайс Redux оновлює локальний стан (див. рис. 3.20).

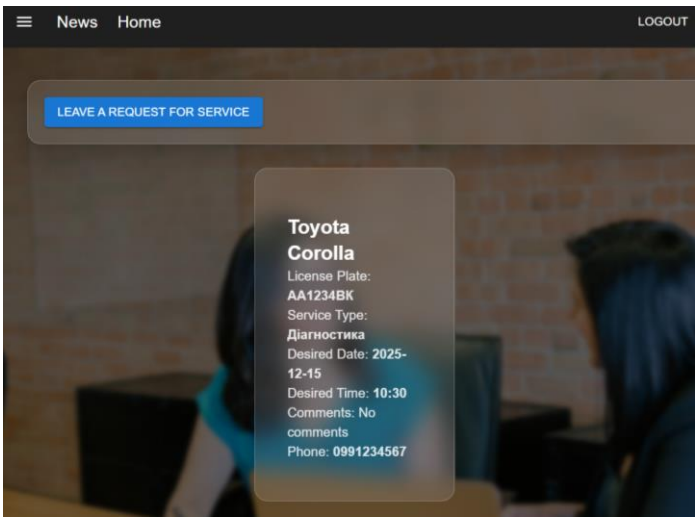


Рисунок 3.19 – Відображення замовлення у React після створення


```
{
  "_id": {
    "$oid": "66478d51f5b7dc3b004f330c"
  },
  "id": "6b2921d5-6c91-4f5d-89d0-6455d2453540",
  "carBrand": " Toyota ",
  "carModel": " Corolla ",
  "licensePlate": " AA1234BK ",
  "serviceType": " Діагностика ",
  "desiredDate": "2025-12-15",
  "desiredTime": "10:30",
  "comments": "",
  "phoneNumber": "0991234567",
  "userId": "07133512-86f9-4072-a550-be198a515b51",
  "updatedAt": {
    "$date": "2025-12-02T18:54:23.101Z"
  }
}
```


Рисунок 3.20 – Новий документ у колекції Orders (MongoDB)

Сценарій 4. Додавання завдання до замовлення. Мета тесту: перевірити можливість створення завдання для конкретного замовлення.

Вхідні дані (див. рис. 3.21):

– заголовок: заміна фільтра;

– orderId: ідентифікатор існуючого замовлення.

Очікуваний результат: завдання додається до списку робіт (див. рис. 3.22) та зберігається у колекції Tasks (див. рис. 3.23).

Фактичний результат: функція addTaskSank() викликає POST-запит /tasks, Redux-стан оновлюється, у списку завдань з’являється новий елемент.

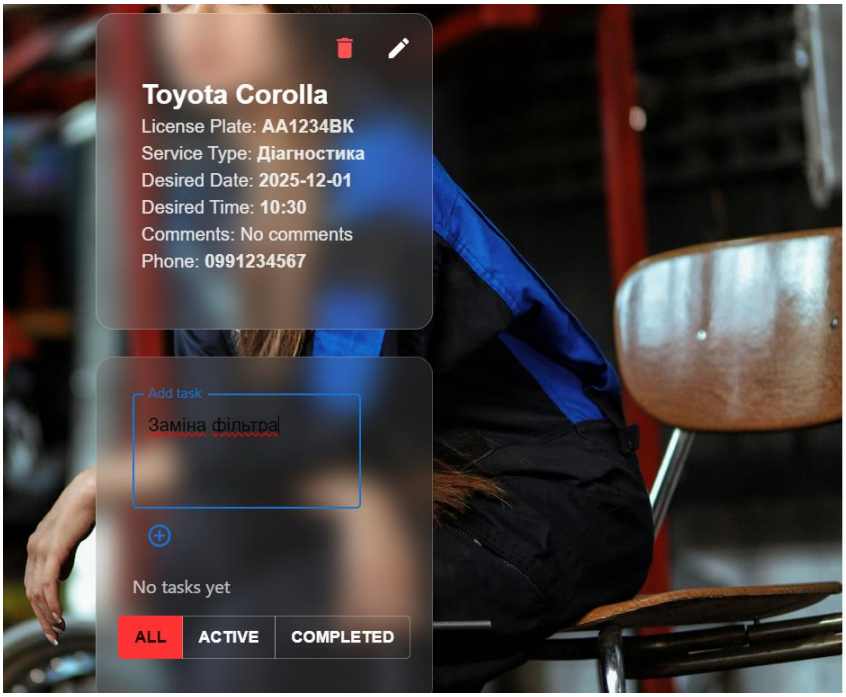


Рисунок 3.21 – Створення завдання у формі

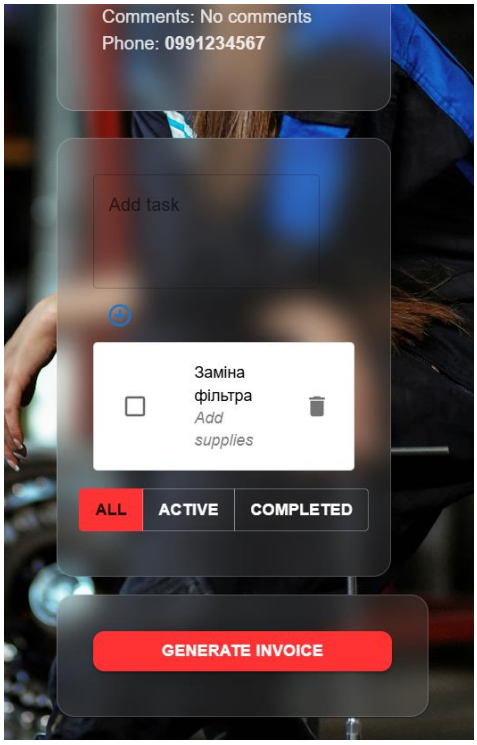


Рисунок 3.22 – Відображення завдання у списку робіт

```
{
  "_id": {
    "$oid": "665d79b355189797eb5c80fd"
  },
  "id": "e7c0b6e0-33f5-42ba-98ee-14b22e9d2e9d",
  "orderId": "6b2921d5-6c91-4f5d-89d0-6455d2453540",
  "title": " Заміна фільтра ",
  "supplies": "Add supplies",
  "suppliesCost": 0,
  "isDone": false,
  "__v": 0
}
```


Рисунок 3.23 – Збережений документ у колекції Tasks

Сценарій 5. Видалення замовлення разом із завданнями. Мета: перевірити коректність комплексного видалення замовлення та пов’язаних завдань.

Очікуваний результат: замовлення видалено з колекції Orders, пов’язані завдання видалено з Tasks.

Фактичний результат: запит DELETE /orders/:id викликає два процеси:

– видалення замовлення (див. рис. 3.24);

– видалення всіх робіт через removeOrderTasksSank() (див. рис. 3.25).

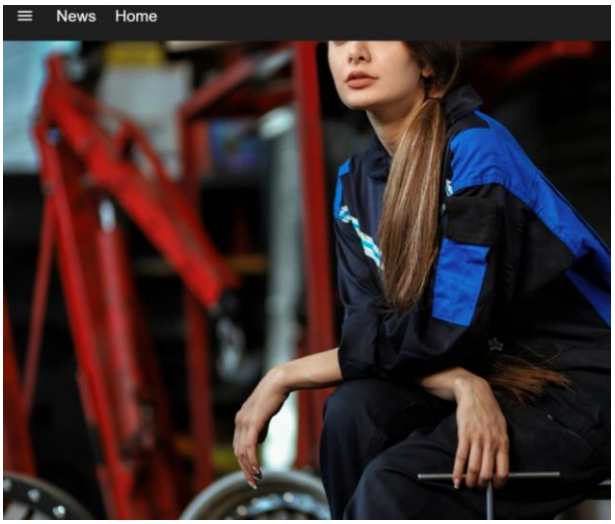


Рисунок 3.24 – Видалення замовлення у фронтенді



Рисунок 3.25 – Порожній список завдань після видалення та відсутність відповідних документів у колекції Tasks

Сценарій 6. Формування рахунку (Invoice). Мета: перевірити коректність створення фінального рахунку на основі замовлення та його завдань.

Очікуваний результат: система генерує об’єкт Invoice, копіює перелік завдань, розраховує суму.

На рисунках 3.26 – 3.28 відповідно наведено інтерфейс створення рахунку, приклад сформованого інвойсу та вигляд документа Invoice у базі даних MongoDB.

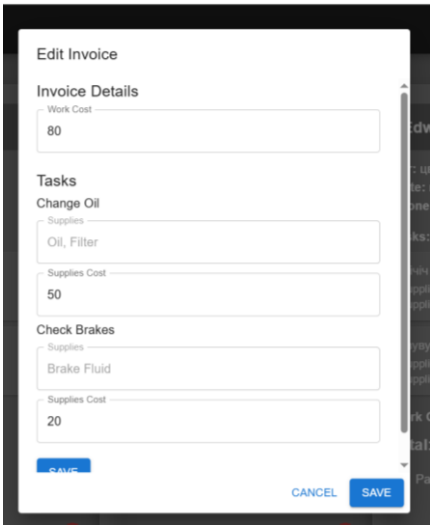


Рисунок 3.26 – Форма створення рахунку

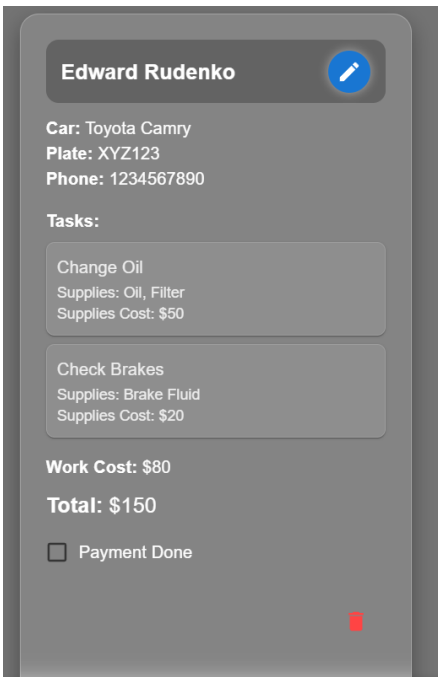


Рисунок 3.27 – Відображення сформованого рахунку



Рисунок 3.28 – Документ Invoice у MongoDB

3.3.1 Модульне тестування (Unit-тестування)

Модульне тестування (unit testing) – це процес перевірки працездатності та коректності роботи окремих програмних модулів, зокрема функцій, компонентів або методів класів. Основна мета модульного тестування полягає

у виявленні помилок на ранніх етапах розроблення, локалізації джерел некоректної поведінки та запобіганні регресіям після внесення змін у код.

Під час розроблення вебдодатку AVTOREPAIR модульні тести були застосовані для перевірки коректності роботи окремих React-компонентів та Redux-ред'юсерів. Юніт-тести створювалися за допомогою середовища Jest та бібліотеки @testing-library/react, що дозволяє тестувати поведінку компонентів аналогічно їх роботі в браузері.

Опис модульного тестування компонента EditInvoiceForm. У межах модульного тестування було перевірено коректність роботи компонента EditInvoiceForm, який відповідає за редагування даних рахунку-фактури (інвойсу) та пов'язаних з ним завдань. Основною метою тестування було підтвердження правильності обробки подій зміни даних, коректного відображення вихідних значень та виклику відповідних callback-функцій, що забезпечують передачу змін до бізнес-логіки застосунку.

Для тестування використовувалося середовище Jest у поєднанні з React Testing Library, що відповідає сучасним підходам до тестування React-компонентів та дозволяє симулювати роботу користувача без втручання у внутрішню реалізацію компонента.

Метою модульного тестування було:

– перевірка коректного рендерингу всіх елементів форми;
– перевірка роботи механізму редагування загальних полів інвойсу (зокрема workCost);
– перевірка коректної передачі подій змін полів завдань (tasks) компоненту-батьку;
– перевірка коректної обробки події натискання кнопки "Save";
– підтвердження відповідності поведінки компонента встановленим функціональним вимогам.

Об'єктом тестування є React-компонент EditInvoiceForm, який включає:

– відображення загальних властивостей інвойсу (вартість роботи, коментарі тощо);

– вбудовані компоненти редагування завдань (EditTaskForm);
– кнопку підтвердження збереження змін;
– виклики зовнішніх колбек-функцій: onChange (зміна полів інвойсу), onTaskChange (зміна даних окремого завдання), onSave (збереження змін).

У модульних тестах було реалізовано **чотири** основні сценарії.

Перевірка коректного відображення форми. Тест підтвердив, що:
– заголовки Invoice Details та Tasks відображаються;
– поле введення Work Cost містить коректне стартове значення;
– у формі присутні дані завдання, зокрема назва Oil Change та значення 200 у полі suppliesCost.

Цей сценарій підтвердив коректність рендерингу компоненту та відповідність UI вихідним даним інвойсу.

Перевірка виклику обробника змін загальних полів інвойсу. Було проведено тест на зміну значення поля workCost. За результатами:
– компонент коректно сприймає подію change;
– функція onChange викликається рівно один раз;
– переданий параметр відповідає події користувача.

Це підтверджує правильність реалізації двостороннього зв’язку між формою та бізнес-логікою.

Перевірка виклику функції onTaskChange. Було змодельовано зміну значення у полі suppliesCost конкретного завдання. Тест визначив, що:
– зміна поля викликає onTaskChange;
– компонент передає коректний ідентифікатор завдання;
– подія містить нове значення, змодельоване у тесті.

Це свідчить про правильну обробку вкладених елементів та коректне делегування відповідальності.

Перевірка виклику onSave. Окремим тестом було перевірено:
– коректність рендерингу кнопки “Save”;
– виклик функції onSave при натисканні.

Тест підтвердив, що користувач може успішно завершити редагування інвойсу та передати зміни до вищого рівня логіки.

Усі тести успішно підтвердили коректність роботи механізмів:

– відображення компоненту;

– обробки подій користувача;

– делегування змін у зовнішні callback-функції;

– збереження модифікованих даних.

Жодних регресій або некоректного поводження компоненту виявлено не було.

Проведене модульне тестування дозволило підтвердити:

– відповідність компонента EditInvoiceForm встановленим функціональним вимогам;

– коректність його взаємодії з підкомпонентами та бізнес-логікою застосунку.

Фрагмент коду тестування наведено на рисунку 3.29. Повний код тестування – у додатку А.

```
test('calls onSave when Save button is clicked', () => {
  const handleSave = jest.fn();
  render(
    <EditInvoiceForm
      invoice={mockInvoice}
      onChange={jest.fn()}
      onTaskChange={jest.fn()}
      onSave={handleSave}
    />
  );
  fireEvent.click(screen.getByText(/Save/i));

  expect(handleSave).toHaveBeenCalledTimes(1);
});
```

Рисунок 3.29 – Тестування EditInvoiceForm, фрагмент

3.3.2 Інтеграційне тестування

Інтеграційне тестування (Integration Testing) – це етап контролю якості програмного забезпечення, під час якого перевіряється взаємодія між окремими модулями, компонентами або підсистемами. На відміну від модульного тестування, що охоплює одиничні функції або класи, інтеграційні тести визначають коректність роботи системи у зв'язці декількох елементів та забезпечують виявлення дефектів, пов'язаних з передаванням даних, сумісністю інтерфейсів та логікою взаємодії.

У межах вебсистеми AVTOREPAIR, реалізованої на MERN-технологіях, інтеграційне тестування виконується для компонентів, що взаємодіють із Redux-станом, REST API та формами введення даних. Зокрема, було проведено тестування механізму створення сервісної заявки користувачем через компонент OrderFormAdd, який викликає Redux-асинхронний thunk addOrderAsync, відправляє дані до API-сервера та оновлює глобальний стан застосунку.

В додатку Б наведено код інтеграційного тесту, який перевіряє взаємодію з: компонентом введення даних, Redux-сховищем, асинхронною логікою створення сервісної заявки, оновленням сторінки зі списком замовлень.

Аналіз проведеного інтеграційного тестування. У процесі тестування було виконано такі основні кроки:відтворено повну роботу інтерфейсу через render() у контексті сховища, перевірено заповнення всіх полів форми користувачем, здійснено симуляцію події submit, зафіксовано виклик асинхронного Redux-методу addOrderAsync, який забезпечує створення нового сервісного замовлення, перевірено коректне передавання структури об'єкта заявки, що генерується у компоненті, підтверджено правильність інтеграції UI-форми, бізнес-логіки та рівня взаємодії з API.

Отже, інтеграційний тест підтверджує, що модулі OrderFormAdd, Redux-логіка та API-взаємодія функціонують узгоджено, забезпечуючи стабільний процес створення сервісних заявок.

3.4 Керівництво користувача

3.4.1 Рівень підготовки користувача

Для ефективної роботи з розробленою інформаційною системою користувач повинен відповідати певним кваліфікаційним вимогам.

Користувач має володіти базовими навичками роботи з персональним комп’ютером, зокрема:

– умінням користуватися клавіатурою та маніпулятором «миша»;
– знаннями щодо роботи з операційною системою на рівні запуску програм, роботи з файлами та каталогами;
– розумінням принципів роботи клієнт-серверних вебзастосунків.

Крім того, користувач повинен мати навички роботи з сучасними веббраузерами: відкриття вкладок, переходу за гіперпосиланнями, заповнення форм, використання навігаційних елементів інтерфейсу.

Для забезпечення коректної роботи в системі рекомендується ознайомитися з Керівництвом користувача, яке містить відомості про функціональні можливості, порядок виконання основних операцій та правила експлуатації сервісу.

3.4.2 Підготовка до роботи

Запуск системи здійснюється через мережу Інтернет за допомогою стандартного веббраузера. Для коректної роботи клієнтської частини рекомендовано використовувати сучасні браузери: Google Chrome, Mozilla Firefox, Microsoft Edge, Opera або Safari.

Система AVTOREPAIR не потребує встановлення додаткового програмного забезпечення. Доступ до сервісу здійснюється через вебадресу: https://autorepair.loc (або фактичний домен застосунку).

Після переходу на сайт користувач автоматично потрапляє на сторінку About.

Групи користувачів системи. Анонімний користувач (гість):

– не має облікового запису або не виконав вхід;
– може переглядати лише сторінку авторизації/реєстрації;
– доступ до функціональних модулів заборонено.

Авторизований користувач (Клієнт / Працівник / Менеджер) після входу отримує доступ до персонального кабінету згідно з роллю:

– Customer (клієнт) – перегляд власних замовлень, записів на ремонт, профілю;
– Worker (майстер) – доступ до списку робіт на день, позначення статусів, перегляд інвойсів;
– Manager – управління телефонними замовленнями, перегляд усіх замовлень клієнтів.

Адміністратор системи:

– управляє обліковими записами, інвойсами, процесами ремонту;
– виконує налаштування, контроль і діагностику системи;
– має повний доступ до всіх модулів.

Прецедент роботи користувача. Основним сценарієм роботи є отримання доступу до особистого кабінету та взаємодія з модулями системи.

Нижче наведено алгоритм роботи:

– користувач відкриває вебсайт;
– система перевіряє наявність дійсного токена автентифікації: якщо токен дійсний – виконується автоматичний вхід; якщо токен недійсний – користувач бачить сторінку входу;
– користувач вводить облікові дані;
– після перевірки даних система визначає роль користувача;
– інтерфейс підлаштовується відповідно до ролі;
– користувач переходить до потрібного розділу за допомогою бічного меню (Navbar);

– користувач виконує дії згідно з роллю перегляд інвойсів, телефонних замовлень, робіт на день тощо.

Виняткові ситуації. Невалідні дані при вході: система підсвічує поля червоним та виводить повідомлення помилки.

Невірний логін або пароль: відображається повідомлення «Невірний логін або пароль».

Прострочений токен: система автоматично виконує вихід та перенаправляє на сторінку входу.

Спроба доступу до забороненого розділу: система перенаправляє користувача на головну сторінку профілю та повідомляє про недостатні права.

3.4.3 Вхід до системи

Після відкриття вебсайту AVTOREPAIR система автоматично завантажує головну сторінку About (див. рис. 3.30), яка містить загальну інформацію про сервіс, структуру та можливості системи.

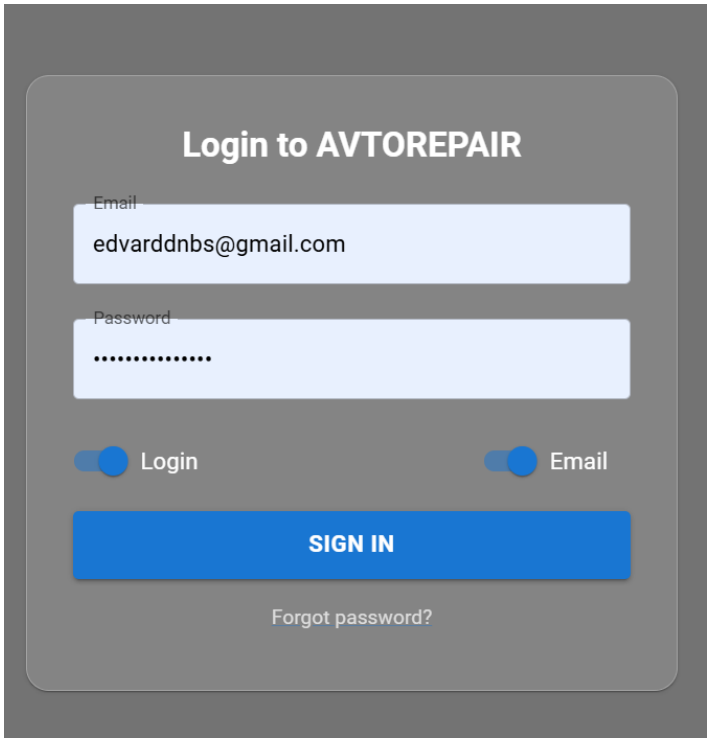


Рисунок 3.30 – Форма входу до системи

Для авторизації користувач переходить до форми входу, де вводить адресу електронної пошти або номер телефону та пароль, після чого натискає кнопку “Login/Register”, ініціюючи перевірку даних та визначення свого статусу в системі.

У випадку коректного введення інформації користувач отримує доступ до персоналізованого інтерфейсу відповідно до своєї ролі в системі (менеджер, працівник або клієнт).

Реакція на некоректні облікові дані. Якщо користувач із вказаним логіном або паролем не існує, система виводить повідомлення (див. рис. 3.31).

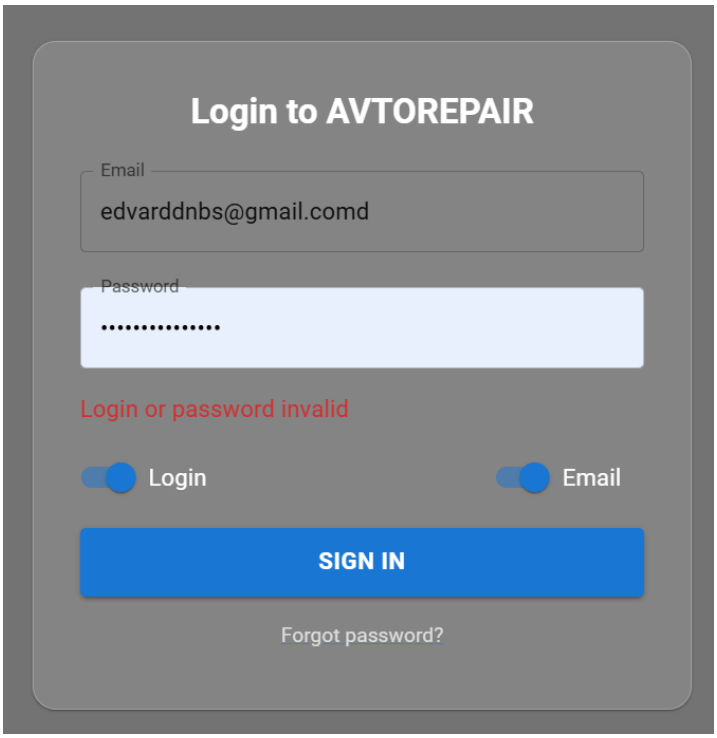


Рисунок 3.31 – Повідомлення про некоректні облікові дані

Після успішної авторизації система визначає роль користувача та завантажує відповідний інтерфейс. Наприклад, для працівника відображається сторінка сьогоднішніх робіт (див. рис. 3.32), а для клієнта замовлення, роботи та інвойси відносно його замовлення та інформація що до його облікового запису. На рисунку 3.33 представлено інтерфейс робочої сторінки користувача, яка автоматично відкривається після успішної авторизації в системі.

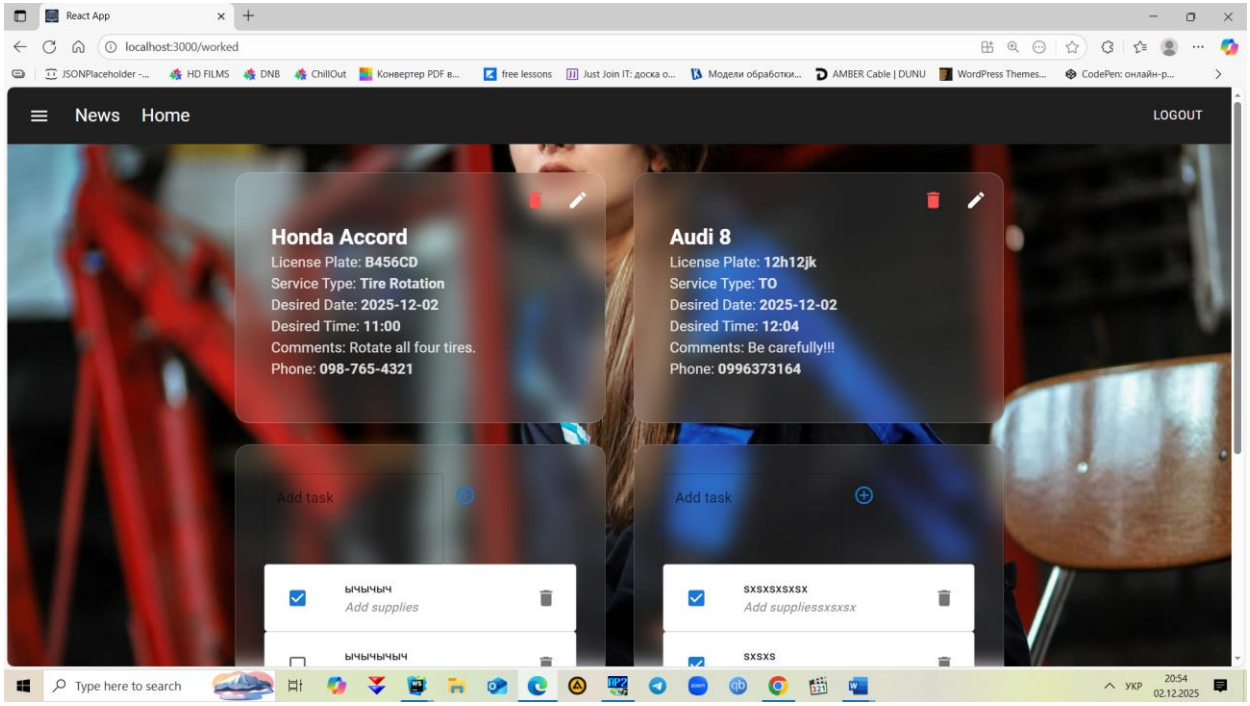


Рисунок 3.32 – Сторінка сьогоднішніх робіт

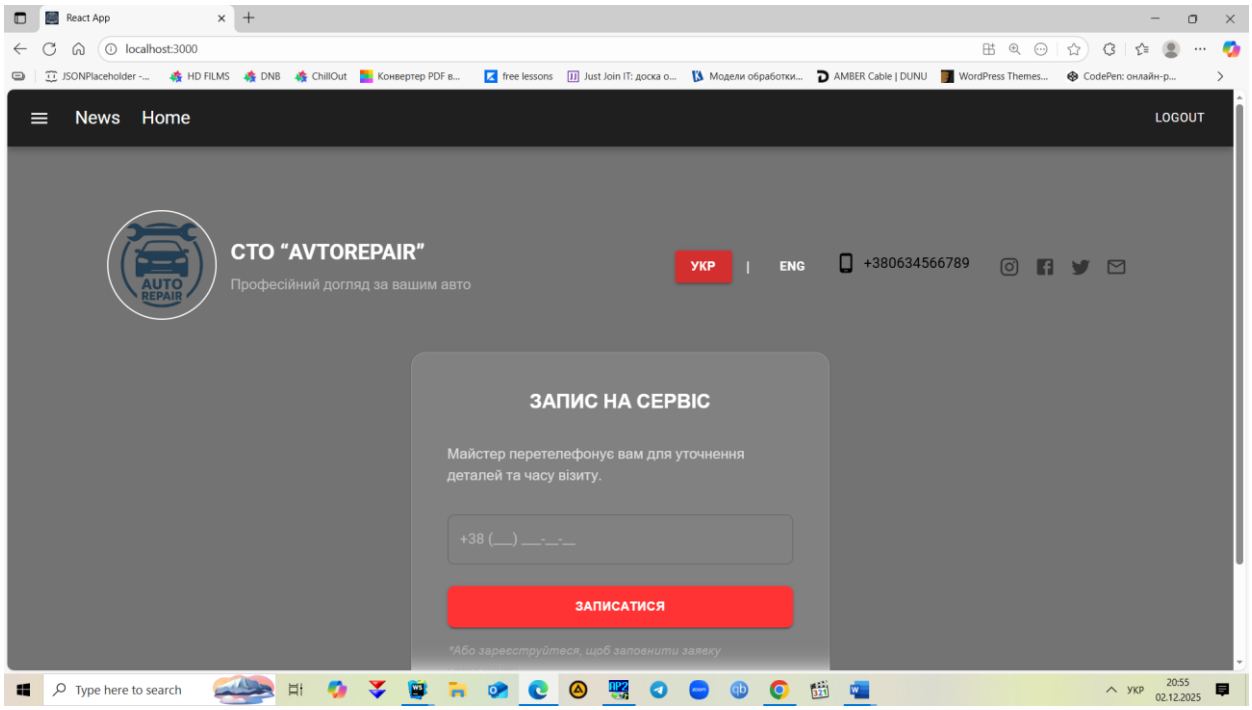


Рисунок 3.33 – Робоча сторінка користувача після входу

ВИСНОВКИ

У процесі виконання дипломної роботи було проведено комплексне дослідження, аналіз та розроблення веборієнтованої інформаційної системи, призначеної для автоматизації бізнес-процесів автосервісу. Система реалізує повний цикл взаємодії між клієнтами, майстрами, менеджерами та адміністраторами, забезпечуючи зручне керування замовленнями, облік виконаних робіт, формування інвойсів, а також підтримку телефонних звернень.

У ході виконання роботи було проаналізовано сучасні підходи до побудови вебсистем, вивчено вимоги до автоматизації діяльності автосервісних підприємств та визначено недоліки існуючих рішень. На основі цього сформовано технічні вимоги, архітектуру та модель даних майбутньої системи. Окрему увагу приділено питанням юзабіліті, безпеки та масштабованості застосунку.

Було розроблено клієнтську частину системи з використанням React, TypeScript, Material UI, що забезпечило високу інтерактивність інтерфейсу, адаптивність і зручність роботи для різних типів користувачів. Застосовано Redux Toolkit для централізованого зберігання стану та керування бізнес-логікою, що дозволило досягти передбачуваності виконання операцій та спростити обробку асинхронних запитів. Реалізовано систему ролей (клієнт, майстер, менеджер, адміністратор), що гарантує коректний розподіл доступів і безпечну роботу в багатокористувацькому середовищі.

Серверна частина, побудована на базі REST-API, забезпечує обробку замовлень, створення інвойсів, управління користувачами та облік виконаних робіт. Впроваджено механізми авторизації та аутентифікації з використанням токенів, що дає змогу підтримувати стійкі сесії та підвищує безпеку доступу.

Особливістю розробленої системи є інтеграція кількох взаємопов’язаних модулів: модуль записів та телефонних замовлень, модуль

формування та редагування інвойсів, панель майстра з переліком завдань на день, панель менеджера для роботи з клієнтськими заявками, особистий кабінет клієнта.

У роботі проведено модульне (unit-testing) та інтеграційне тестування, що дозволило оцінити якість роботи окремих компонентів (наприклад, EditInvoiceForm) та взаємодію між ними. Застосування Jest і React Testing Library забезпечило верифікацію коректності критичних функцій, зокрема створення, оновлення та обробки інвойсів, відображення інформації про замовлення та правильності відтворення інтерфейсу. Проведене тестування підтвердило стабільність роботи системи та відповідність функціональних можливостей технічним вимогам.

Результатом виконання дипломного проєкту стала повнофункціональна інформаційна система, яка:

- автоматизує ключові процеси автосервісу;
- мінімізує людський фактор під час обробки замовлень;
- скорочує час взаємодії з клієнтами;
- забезпечує прозорий облік виконаних робіт;
- формує інвойси та дозволяє зберігати історію обслуговування;
- підвищує ефективність управління та прийняття рішень.

Таким чином, поставлені в дипломній роботі завдання повністю виконано. Розроблена система відповідає вимогам сучасних вебтехнологій, може бути впроваджена в роботу малого та середнього автосервісного підприємства, а також має потенціал для подальшого розширення та інтеграції з іншими сервісами (SMS-нагадування, онлайн-запис, бухгалтерські системи, складський облік тощо).

Розроблений програмний продукт є ефективним інструментом цифрової трансформації діяльності автосервісу та може бути основою для подальших наукових і практичних досліджень у галузі автоматизації бізнес-процесів.

ПЕРЕЛІК ПОСИЛАНЬ

1. RemOnline. URL: https://remonline.ua/ (дата звернення: 18.05.2025).
2. Ulteama – CRM для автосервісу. *Ulteama*. URL: https://ulteama.com/uk/ (дата звернення: 18.05.2025).
3. CARBOOK – Програмне забезпечення для СТО. *Carbook Mobi*. URL: https://carbook.mobi/ (дата звернення: 18.05.2025).
4. Менеджер автосервісу. *E-Soft Auto*. URL: https://www.esoft-auto.com (дата звернення: 18.05.2025).
5. What is the MERN Stack? Guide & Examples. *Oracle.* URL: https://www.oracle.com/ba/database/mern-stack/ (дата звернення: 25.05.2025).
6. What is JavaScript? *Noble Desktop*. URL: https://www.nobledesktop.com/learn/mern-stack/what-is-mern-stack (дата звернення: 25.05.2025).
7. Марчук Д. В. Розробка веб-платформи для створення та ведення блогів з використанням React, Node.js та MongoDB : кваліфікаційна робота бакалавра : спец. 121 «Інженерія програмного забезпечення». Тернопіль : ТНТУ ім. Івана Пулюя, 2024. 47 с. URL: https://elartu.tntu.edu.ua/bitstream/lib/45491/1/dyplom_Marchuk_2024.pdf (дата звернення: 25.05.2025).
8. Шальнєв О. В. Веб-застосунок для продажу комп'ютерної техніки : кваліфікаційна робота бакалавра : спец. 121 «Інженерія програмного забезпечення». Миколаїв : Чорноморський національний університет імені Петра Могили, 2023. URL: https://krs.chmnu.edu.ua/jspui/bitstream/123456789/2976/1/Шальнєв%20Олександр%20БКР.pdf (дата звернення: 25.05.2025).
9. Gadhavi M. Full-Stack vs MEAN Stack vs MERN Stack: The Right Technology Stack for You in 2025. Radixweb, 2025. URL: https://radixweb.com/blog/full-stack-vs-mean-stack-vs-mern-stack-development (дата звернення: 25.08.2025).

10. EasyShiksha. Веб-розробка Full Stack за допомогою MERN STACK : безкоштовний онлайн-курс. URL: https://easyshiksha.com/ (дата звернення: 25.05.2025).

11. React Official Documentation. URL: https://react.dev (дата звернення: 25.05.2025).

12. Mozilla Developer Network. JavaScript Guide. URL: https://developer.mozilla.org/ (дата звернення: 25.05.2025).

13. Node.js Official Documentation. URL: https://nodejs.org (дата звернення: 25.05.2025).

14. MongoDB Manual. URL: https://www.mongodb.com/docs/ (дата звернення: 25.05.2025).

ДОДАТОК А

Код тестування EditInvoiceForm.tsx

```
// src/tests/EditInvoiceForm.test.tsx
import React from 'react';
import { render, screen, fireEvent } from '@testing-library/react';
import { EditInvoiceForm } from '../EditInvoiceForm';
import { Invoice } from '../state/invoicesSlice';

const mockInvoice: Invoice = {
  id: '1',
  orderId: '101',
  firstName: 'John',
  lastName: 'Doe',
  carBrand: 'BMW',
  carModel: 'X5',
  licensePlate: 'AA1234BB',
  serviceType: 'Engine Repair',
  desiredDate: '2024-12-12',
  desiredTime: '14:00',
  comments: '',
  phoneNumber: '+380991234567',
  workCost: 500,
  totalCost: 700,
  paymentDone: false,
  tasks: [
    {
      id: 't1',
      title: 'Oil Change',
      supplies: 'Oil 5W30',
      suppliesCost: 200,
    }
  ]
};

describe('EditInvoiceForm', () => {

  test('renders invoice fields', () => {
    render(
```

```
      <EditInvoiceForm
        invoice={mockInvoice}
        onChange={jest.fn()}
        onTaskChange={jest.fn()}
        onSave={jest.fn()}
      />
    );

    // Заголовки
    expect(screen.getByText(/Invoice Details/i)).toBeInTheDocument();
    expect(screen.getByText(/Tasks/i)).toBeInTheDocument();

    // Поле WorkCost
    expect(screen.getByLabelText(/Work Cost/i)).toHaveValue(500);

    // Тайтл таску рендериться як <Typography>, не input
    expect(screen.getByText('Oil Change')).toBeInTheDocument();

    // Поле SuppliesCost існує
    expect(screen.getByDisplayValue('200')).toBeInTheDocument();
});

test('calls onChange when Work Cost is changed', () => {
    const handleChange = jest.fn();

    render(
      <EditInvoiceForm
        invoice={mockInvoice}
        onChange={handleChange}
        onTaskChange={jest.fn()}
        onSave={jest.fn()}
      />
    );

    const input = screen.getByLabelText(/Work Cost/i);

    fireEvent.change(input, { target: { value: '999' } });

    expect(handleChange).toHaveBeenCalledTimes(1);
});

test('calls onTaskChange when Supplies Cost is edited', () => {
```

```
    const handleTaskChange = jest.fn();

    render(
      <EditInvoiceForm
        invoice={mockInvoice}
        onChange={jest.fn()}
        onTaskChange={handleTaskChange}
        onSave={jest.fn()}
      />
    );

    // Шукаємо поле Supplies Cost (воно editable)
    const input = screen.getByDisplayValue('200');

    fireEvent.change(input, { target: { value: '250' } });

    expect(handleTaskChange).toHaveBeenCalledTimes(1);
    expect(handleTaskChange).toHaveBeenCalledWith(
      't1',
      expect.any(Object)
    );
  });

  test('calls onSave when Save button is clicked', () => {
    const handleSave = jest.fn();

    render(
      <EditInvoiceForm
        invoice={mockInvoice}
        onChange={jest.fn()}
        onTaskChange={jest.fn()}
        onSave={handleSave}
      />
    );

    fireEvent.click(screen.getByText(/Save/i));

    expect(handleSave).toHaveBeenCalledTimes(1);
  });
});
```

ДОДАТОК Б

Код тестування CreateOrder.tsx

```
// src/tests/CreateOrder.integration.test.tsx
import React from 'react';
import { render, fireEvent, screen } from '@testing-library/react';
import { Provider } from 'react-redux';
import { store } from '../state/store';
import { OrderFormAdd } from '../OrderFormAdd';
import { addOrderAsync } from '../state/ordersSlice';

jest.mock('../state/ordersSlice', () => ({
  ...jest.requireActual('../state/ordersSlice'),
  addOrderAsync: jest.fn(() => ({ type: 'orders/addOrderAsync' })),
}));

describe('Integration: Create Service Order', () => {
  test('creates a new order successfully', () => {
    render(
      <Provider store={store}>
        <OrderFormAdd title="Leave a request for service" />
      </Provider>
    );

    // Введення даних у форму
    fireEvent.change(screen.getByLabelText(/Brand/i), { target: { value: 'BMW' }});
    fireEvent.change(screen.getByLabelText(/Model/i), { target: { value: 'X5' }});
    fireEvent.change(screen.getByLabelText(/License Plate/i), { target: { value: 'AA1234BB' }});

    // Надсилання заявки
    fireEvent.click(screen.getByText(/Create/i));

    // Перевірка виклику Redux-thunk
    expect(addOrderAsync).toHaveBeenCalledTimes(1);
    expect(addOrderAsync).toHaveBeenCalledWith(expect.objectContaining({
      carBrand: 'BMW',
      carModel: 'X5',
      licensePlate: 'AA1234BB',
    }));
```