

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка системи аналізу автосервісу для підвищення
якості обслуговування»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень. Використання
ідей, результатів і текстів інших авторів мають посилання
на відповідне джерело*

(підпис) Богдан СИЧ

Виконав: здобувач вищої освіти групи ПД-44

Богдан СИЧ

Керівник:
канд.техн.наук

Юрій ЗАДОНЦЕВ

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

«_____» _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____ Сичу Богдану Олеговичу _____

1. Тема кваліфікаційної роботи: «Розробка системи аналізу автосервісу для підвищення якості обслуговування».

керівник кваліфікаційної роботи канд.техн.наук Юрій ЗАДОНЦЕВ,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про процеси управління роботою автосервісу, організацію обліку заявок на ремонт, клієнтів, автомобілів, майстрів і виконаних робіт, підходи до контролю якості обслуговування, принципи побудови аналітичних модулів для оцінювання стану сервісу, особливості розробки клієнт-серверних web-застосунків, технічна документація щодо використання технологій Java, Spring Boot, Spring Security, JWT, Spring Data JPA, PostgreSQL, Liquibase, React, Vite, TypeScript, Axios, React Router, REST API, Docker та Git.

4. Зміст розрахунково-пояснювальної записки:

1. Огляд та аналіз предметної галузі управління роботою автосервісу.
2. Аналіз існуючих програмних засобів для автоматизації автосервісів.
3. Огляд засобів реалізації web-системи ServiceControl.
4. Проектування структури, бази даних та основних модулів системи.
5. Програмна реалізація та опис функціонування web-системи.
6. Тестування web-системи ServiceControl.
7. Апробація результатів дослідження.
8. Перелік графічного матеріалу: презентація.

3. Вимоги до web-застосунку:

1. Вимоги до web-системи ServiceControl.
2. Логічна структура системи ServiceControl.
3. Діаграма варіантів використання системи ServiceControl.
4. Блок-схема процесу обробки заявки на ремонт.
5. Логіка формування індексу Service Health Radar.
6. Структура бази даних web-системи.
7. Архітектура web-системи ServiceControl.
8. Схема взаємодії frontend, backend та бази даних через REST API.
9. ER-діаграма бази даних системи ServiceControl.
10. Діаграма класів основних модулів системи.
11. Схема та результати тестування програмного забезпечення.
12. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури з теми управління роботою автосервісу та автоматизації сервісних процесів	23.02–02.03.2026	
2	Аналіз існуючих аналогів програмного забезпечення для автоматизації автосервісів і контролю заявок	03.03–09.03.2026	
3	Аналіз предметної галузі управління автосервісом та визначення вимог до web-системи ServiceControl	10.03–17.03.2026	
4	Огляд засобів реалізації web-системи ServiceControl	18.03–24.03.2026	

5	Проектування web-системи ServiceControl, структури системи, бази даних та UML-діаграм	25.03–07.04.2026	
6	Програмна реалізація серверної та клієнтської частини web-системи	08.04–25.04.2026	
7	Реалізація аналітичного модуля Service Health Radar, Dashboard та модуля звітів	22.04–29.04.2026	
8	Тестування web-системи ServiceControl та оформлення результатів тестування	30.04–06.05.2026	
9	Оформлення роботи: вступ, висновки, реферат, список джерел та додатки	07.05–12.05.2026	
10	Розробка демонстраційних матеріалів і презентації	13.05–17.05.2026	
11	Попередній захист роботи	18.05–22.05.2026	

Здобувач вищої освіти

(підпис)

Богдан СИЧ

Керівник

кваліфікаційної роботи

(підпис)

Юрій ЗАДОНЦЕВ

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 65 стор., 4 табл., 23 рис., 23 джерел.

Мета роботи – спрощення аналізу роботи автосервісу за рахунок розробки web-системи ServiceControl для централізованого обліку заявок, клієнтів, автомобілів, майстрів та виконаних робіт.

Об'єкт дослідження – процеси управління заявками, ремонтними роботами та якістю роботи автосервісу.

Предмет дослідження – засоби, методи та технології реалізації web-системи ServiceControl для автоматизації аналізу роботи автосервісу та показників якості обслуговування.

Короткий зміст роботи: у кваліфікаційній роботі проведено аналіз предметної галузі управління роботою автосервісу та визначено основні проблеми, пов'язані з розрізненням зберіганням даних, складністю контролю строків виконання заявок, нерівномірним навантаженням майстрів та відсутністю комплексної аналітики якості обслуговування. Проаналізовано існуючі програмні засоби для автоматизації автосервісів, зокрема AutoLeap, Shopmonkey, AutoFluent та ARI Auto Repair Software, визначено їх переваги та недоліки.

У роботі сформовано функціональні та нефункціональні вимоги до системи ServiceControl, обґрунтовано вибір засобів реалізації клієнтської частини, серверної частини та бази даних. Для реалізації web-системи використано Java, Spring Boot, Spring Security, Spring Data JPA, PostgreSQL, React, TypeScript, Vite, Tailwind CSS, REST API, Docker та Git.

Спроектовано логічну структуру системи, рольову модель доступу, структуру бази даних, ER-діаграму, діаграму варіантів використання, блок-схему процесу обробки заявки на ремонт та схему взаємодії frontend, backend і бази даних. Реалізовано основні модулі системи: авторизацію та реєстрацію користувачів,

профіль користувача, управління клієнтами, автомобілями, майстрами, користувачами, заявками на ремонт, виконаними роботами, відгуками, історією подій та звітами.

Особливістю розробленої системи є аналітичний модуль Service Health Radar, який оцінює загальний стан автосервісу на основі кількості прострочених заявок, завантаженості майстрів, заявок без призначеного виконавця, середньої оцінки клієнтів і проблемних категорій ремонтних робіт. Проведено тестування системи, перевірено коректність роботи основних модулів, рольового доступу, Dashboard та Service Health Radar.

КЛЮЧОВІ СЛОВА: АВТОСЕРВІС, SERVICECONTROL, WEB-СИСТЕМА, ЗАЯВКА НА РЕМОНТ, DASHBOARD, SERVICE HEALTH RADAR, SPRING BOOT, REACT, POSTGRESQL.

ЗМІСТ

РЕФЕРАТ	7
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	11
ВСТУП.....	13
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ РІШЕНЬ ДЛЯ УПРАВЛІННЯ АВТОСЕРВІСОМ.....	16
1.1 Загальна характеристика процесів роботи автосервісу	16
1.2 Проблеми автоматизації та контролю якості обслуговування.....	18
1.3 Аналіз існуючих програмних засобів	21
1.4 Обґрунтування доцільності розробки системи ServiceControl.....	24
1.5 Визначення вимог до програмного забезпечення	27
2 ЗАСОБИ РЕАЛІЗАЦІЇ СИСТЕМИ SERVICECONTROL.....	30
2.1 Обґрунтування вибору клієнт-серверної архітектури.....	30
2.2 Засоби реалізації серверної частини	32
2.3 Засоби реалізації клієнтської частини	34
2.4 Засоби збереження та обробки даних	36
2.5 Додаткові інструменти розробки та розгортання.....	39
3 ПРОЄКТУВАННЯ СИСТЕМИ SERVICECONTROL	41
3.1 Загальна логічна структура системи.....	41
3.2 Проєктування ролей користувачів та прав доступу	44
3.3 Проєктування бази даних.....	46
3.4 Проєктування процесу роботи із заявкою на ремонт.....	49
3.5 Проєктування аналітичного модуля Service Health Radar	52
3.6 Проєктування API та взаємодії компонентів системи	54

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ SERVICECONTROL	56
4.1 Реалізація авторизації, реєстрації та рольової моделі.....	56
4.2 Реалізація головної сторінки Dashboard та Service Health Radar	58
4.3 Реалізація модуля заявок та виконаних робіт	61
4.4 Реалізація довідників клієнтів, автомобілів і майстрів.....	65
4.5 Реалізація адміністративного модуля користувачів.....	67
5 ТЕСТУВАННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ РОЗРОБКИ.....	68
5.1 Мета та підхід до тестування системи.....	68
5.2 Тестування основних функціональних модулів	70
5.3 Перевірка рольового доступу	73
5.4 Перевірка роботи аналітичного модуля Service Health Radar	74
5.5 Оцінка відповідності системи поставленим вимогам	76
ВИСНОВКИ.....	78
ПЕРЕЛІК ПОСИЛАНЬ.....	80
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ.....	82
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ.....	90

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – інтерфейс прикладного програмування, що використовується для взаємодії між програмними компонентами системи.

REST – архітектурний стиль організації обміну даними між клієнтською та серверною частинами web-застосунку.

HTTP – протокол передавання даних у мережі Інтернет.

HTTPS – захищена версія протоколу HTTP, що забезпечує безпечний обмін даними між клієнтом і сервером.

JSON – текстовий формат обміну даними, який використовується для передавання інформації між frontend- і backend-частинами системи.

JavaScript – мова програмування, що використовується для створення інтерактивної поведінки web-сторінок.

TypeScript – мова програмування, що розширює JavaScript за рахунок статичної типізації.

React – бібліотека JavaScript для створення компонентного користувацького інтерфейсу.

Vite – інструмент для швидкого створення, запуску та збирання frontend-застосунків.

Axios – бібліотека для виконання HTTP-запитів із клієнтської частини web-застосунку.

React Router – бібліотека для організації маршрутизації між сторінками React-застосунку.

Java – об'єктно-орієнтована мова програмування, що використовується для реалізації серверної частини застосунку.

Spring – фреймворк для створення серверних Java-застосунків.

Spring Boot – інструмент екосистеми Spring, що спрощує створення та запуск серверних web-застосунків.

Spring Security – модуль Spring для реалізації автентифікації, авторизації та захисту доступу до ресурсів системи.

JWT – формат токена, що використовується для безпечної передачі даних авторизації між клієнтом і сервером.

JPA – специфікація Java для роботи з реляційними базами даних через об'єктну модель.

Hibernate – фреймворк об'єктно-реляційного відображення, який забезпечує зв'язок між Java-класами та таблицями бази даних.

PostgreSQL – об'єктно-реляційна система керування базами даних.

Liquibase – інструмент для керування змінами структури бази даних за допомогою міграцій.

SPA – односторінковий web-застосунок, у якому основна взаємодія з користувачем відбувається без повного перезавантаження сторінки.

IDE – інтегроване середовище розробки програмного забезпечення.

Git – система контролю версій, що використовується для відстеження змін у програмному коді.

GitHub – web-сервіс для зберігання репозиторіїв, спільної роботи з кодом та контролю версій.

Docker – платформа контейнеризації, що використовується для запуску компонентів системи в ізольованому середовищі.

ServiceControl – web-система управління та аналізу роботи автосервісу.

Service Health Radar – аналітичний модуль системи ServiceControl, який оцінює загальний стан автосервісу на основі поточних показників.

Dashboard – головна аналітична сторінка системи, на якій відображаються ключові показники роботи автосервісу.

VIN – унікальний ідентифікаційний номер транспортного засобу.

PDF – формат електронного документа, який може використовуватися для експорту звітів системи.

ВСТУП

У сучасних умовах цифровізації бізнес-процесів усе більшого значення набувають інформаційні системи, які дозволяють підприємствам автоматизувати щоденну роботу, зменшити кількість ручних операцій, підвищити точність обліку та швидко отримувати аналітичну інформацію для прийняття управлінських рішень. Особливо актуальним це є для сфери автосервісного обслуговування, де якість роботи залежить не лише від технічного виконання ремонту, а й від організації заявок, контролю строків, завантаженості майстрів, комунікації з клієнтами та аналізу результатів виконаних робіт [1; 4].

У зв'язку з цим актуальним є створення web-системи, яка поєднує облік основних процесів автосервісу та аналітичну оцінку його роботи. Така система повинна дозволяти вести базу клієнтів, автомобілів, майстрів, заявок, виконаних робіт і відгуків, а також формувати показники, які допомагають адміністрації виявляти проблемні зони. На відміну від звичайного електронного журналу заявок, програмне забезпечення має не лише зберігати дані, а й перетворювати їх на корисну управлінську інформацію [1; 3; 6].

Актуальність теми кваліфікаційної роботи зумовлена необхідністю розробки зручної системи управління та аналізу роботи автосервісу, яка дозволяє централізовано обліковувати заявки, контролювати роботу майстрів, фіксувати виконані роботи, враховувати відгуки клієнтів і формувати інтегральний показник стану сервісу. Розроблювана система ServiceControl орієнтована на малі та середні автосервіси, яким потрібен зрозумілий інструмент для щоденної роботи та аналізу якості обслуговування [2; 4; 6].

Особливою функціональною можливістю системи є аналітичний модуль Service Health Radar. Він оцінює загальний стан автосервісу на основі поточних даних: кількості прострочених заявок, заявок без призначеного майстра, завантаженості персоналу, середньої оцінки клієнтів, тривалості ремонту та проблемних категорій робіт. На основі цих показників формується інтегральний індекс стану сервісу та список рекомендацій для адміністратора. Завдяки цьому

ServiceControl виконує не лише облікову, а й аналітичну функцію, що робить систему інструментом підтримки управлінських рішень [1; 3; 6].

Мета роботи – спрощення аналізу роботи автосервісу за рахунок розробки web-системи ServiceControl для централізованого обліку заявок, клієнтів, автомобілів, майстрів та виконаних робіт.

Об'єкт дослідження – процеси управління заявками, ремонтними роботами та якістю роботи автосервісу.

Предмет дослідження – засоби, методи та технології реалізації web-системи ServiceControl для автоматизації аналізу роботи автосервісу та показників якості обслуговування.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати предметну галузь управління автосервісом та визначити проблеми, що впливають на якість обслуговування клієнтів.
2. Дослідити існуючі програмні засоби для автоматизації автосервісів, їх можливості щодо контролю заявок, роботи майстрів та оцінювання сервісу.
3. Сформулювати функціональні та нефункціональні вимоги до web-системи ServiceControl.
4. Дослідити та на основі аналізу обґрунтувати вибір технологій для реалізації клієнтської частини, серверної частини, бази даних та розгортання системи.
5. Спроекувати логічну структуру ServiceControl, рольову модель доступу, базу даних та основні алгоритми аналізу показників роботи автосервісу.
6. Реалізувати основні модулі системи: авторизацію, профіль користувача, управління клієнтами, автомобілями, майстрами та заявками.
7. Розробити механізм фіксації виконаних робіт, відгуків клієнтів, історії подій заявки та показників якості обслуговування
8. Реалізувати аналітичний модуль Service Health Radar для оцінювання стану автосервісу.
9. Провести тестування web-системи ServiceControl та проаналізувати результати перевірки основних функцій.

Методи дослідження. У бакалаврській роботі використовуються такі методи: аналіз предметної галузі, порівняльний аналіз існуючих програмних засобів для автоматизації автосервісів, об'єктно-орієнтоване проєктування, UML-моделювання, проєктування структури реляційної бази даних, розробка клієнт-серверного web-застосунку, реалізація REST API, тестування програмного забезпечення та аналіз результатів роботи системи [7; 8; 9].

Наукова новизна одержаних результатів полягає у розробці структури web-системи ServiceControl, яка поєднує централізований облік роботи автосервісу з аналітичним модулем Service Health Radar. На відміну від звичайних систем ведення заявок, запропоноване рішення дозволяє оцінювати стан сервісу за сукупністю показників: простроченими заявками, завантаженістю майстрів, заявками без виконавця, відгуками клієнтів і проблемними категоріями робіт [1; 3; 6].

Практична значущість одержаних результатів полягає в тому, що розроблена web-система може бути використана малими та середніми автосервісами для централізованого обліку заявок, контролю роботи майстрів, аналізу якості обслуговування та підтримки управлінських рішень. У майбутньому систему можна розширити додаванням модуля онлайн-запису клієнтів, складського обліку запчастин, інтеграції з платіжними сервісами, мобільного застосунку для майстрів, розширеної фінансової аналітики та прогнозування ризику затримки заявки [1; 2; 4; 6].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ РІШЕНЬ ДЛЯ УПРАВЛІННЯ АВТОСЕРВІСОМ

1.1 Загальна характеристика процесів роботи автосервісу

Предметною галуззю кваліфікаційної роботи є процеси управління, обліку та аналізу роботи автосервісу з метою підвищення якості обслуговування клієнтів. Сучасний автосервіс виконує не лише ремонтні або діагностичні роботи, а й забезпечує взаємодію з клієнтами, ведення історії звернень, контроль завантаженості майстрів, облік виконаних робіт, аналіз строків виконання заявок, формування вартості послуг та оцінювання рівня задоволеності клієнтів [1; 2; 4; 6].

Автосервіс як підприємство має низку взаємопов'язаних бізнес-процесів. До них належать приймання заявки від клієнта, реєстрація транспортного засобу, попередній опис проблеми, призначення відповідального майстра, виконання ремонтних робіт, контроль строків виконання, фіксація витраченого часу, формування загальної вартості ремонту, закриття заявки та отримання відгуку від клієнта. Якщо ці процеси виконуються без єдиної інформаційної системи, дані часто зберігаються розрізнено: частина інформації ведеться в паперових журналах, частина — в електронних таблицях, месенджерах або окремих програмах [1; 2; 4; 6].

У таких умовах керівнику або адміністратору складно швидко оцінити реальний стан роботи автосервісу. Наприклад, не завжди зрозуміло, які заявки вже прострочені, які майстри перевантажені, які види робіт найчастіше спричиняють затримки, які клієнти залишають негативні відгуки та які процеси потребують першочергового поліпшення. Це створює потребу у програмній системі, яка не лише зберігає інформацію, а й дозволяє аналізувати її для прийняття управлінських рішень [1; 2; 4; 6].

Основними учасниками процесу роботи автосервісу є клієнт, менеджер, майстер, адміністратор або власник сервісу. Клієнт звертається до автосервісу з

проблемою або запитом на технічне обслуговування автомобіля. Менеджер створює заявку, вносить інформацію про клієнта та автомобіль, визначає тип ремонту, пріоритет і плановий строк завершення. Майстер отримує призначену заявку, виконує ремонтні роботи, фіксує витрачений час, описує виконані дії та додає коментарі. Адміністратор або власник сервісу контролює загальну роботу підприємства, аналізує завантаженість персоналу, строки виконання робіт, фінансові показники та відгуки клієнтів [1; 2; 4; 6].



Рис. 1.1 Загальна схема бізнес-процесу обслуговування клієнта в автосервісі

Особливістю предметної галузі є те, що якість обслуговування в автосервісі залежить не лише від технічного виконання ремонту. Важливими також є швидкість приймання заявки, точність внесення даних, зрозумілість комунікації з клієнтом, дотримання строків, прозорість вартості послуг, правильний розподіл робіт між майстрами та можливість отримати зворотний зв'язок після завершення ремонту [1; 2; 4; 6].

До основних показників, які можуть використовуватися для аналізу роботи автосервісу, належать:

1. кількість активних заявок;
2. кількість завершених заявок;
3. кількість прострочених заявок;
4. середній час виконання ремонту;
5. завантаженість майстрів;
6. кількість заявок без призначеного майстра;
7. загальна вартість виконаних робіт;
8. середня оцінка клієнтів;

9. кількість негативних відгуків;

Використання таких показників дозволяє виявляти слабкі місця в роботі автосервісу. Наприклад, якщо певний майстер має занадто багато активних заявок, це може свідчити про нерівномірний розподіл навантаження. Якщо певна категорія робіт часто завершується із затримкою, це може вказувати на нестачу обладнання, складність робіт або недостатню кваліфікацію персоналу. Якщо середня оцінка клієнтів знижується, адміністрація може швидко відреагувати та проаналізувати причини погіршення якості обслуговування [1; 2; 4; 6].

Таким чином, для ефективної роботи автосервісу потрібна система, яка об'єднує облік, контроль і аналітику. Облікова частина забезпечує збереження даних про клієнтів, автомобілі, заявки, майстрів, виконані роботи та відгуки. Аналітична частина дозволяє обробляти накопичені дані, формувати показники стану сервісу, виявляти проблемні зони та надавати адміністрації інформацію для прийняття рішень [1; 2; 4; 6].

1.2 Проблеми автоматизації та контролю якості обслуговування

У багатьох малих і середніх автосервісах управління роботою все ще частково виконується вручну. Заявки можуть фіксуватися в паперовому журналі, інформація про клієнтів — зберігатися в телефоні менеджера, записи про виконані роботи — передаватися усно або через месенджери, а фінальні розрахунки — вестися в окремих таблицях. Такий підхід може працювати при невеликій кількості клієнтів, однак зі збільшенням кількості заявок він створює значні організаційні проблеми [1; 2; 4; 6].

Однією з основних проблем є розрізненість даних. Коли інформація про клієнтів, автомобілі, заявки, майстрів і виконані роботи зберігається в різних місцях, адміністратору складно швидко отримати повну картину по конкретному ремонту. Наприклад, для аналізу однієї заявки потрібно окремо знайти дані про клієнта, автомобіль, призначеного майстра, список виконаних робіт, вартість

ремонту та відгук. Це збільшує час обробки інформації та підвищує ризик помилок [1; 2; 4; 6].

Другою проблемою є відсутність прозорого контролю строків виконання заявок. Якщо система не відстежує планову дату завершення ремонту та фактичний стан заявки, менеджер або адміністратор можуть не помітити затримку вчасно. У результаті клієнт може отримати автомобіль пізніше очікуваного строку, що негативно впливає на рівень довіри до сервісу [1; 2; 4; 6].

Третьою важливою проблемою є нерівномірне навантаження майстрів. Без автоматизованого перегляду активних заявок складно оцінити, хто з майстрів перевантажений, а хто має вільний робочий ресурс. Через це частина працівників може отримувати надто багато складних заявок, тоді як інші залишаються менш завантаженими. Це впливає і на строки виконання робіт, і на якість обслуговування клієнтів [1; 2; 4; 6].

Також значною проблемою є відсутність повноцінної аналітики. Звичайний облік заявок не дає відповіді на питання, які саме процеси в автосервісі є проблемними. Адміністрації важливо бачити не тільки кількість заявок, а й динаміку їх виконання, проблемні категорії ремонтів, середню оцінку клієнтів, кількість прострочених робіт, ефективність майстрів і загальний стан сервісу. Без цього управлінські рішення часто приймаються інтуїтивно, а не на основі реальних даних [1; 2; 4; 6].

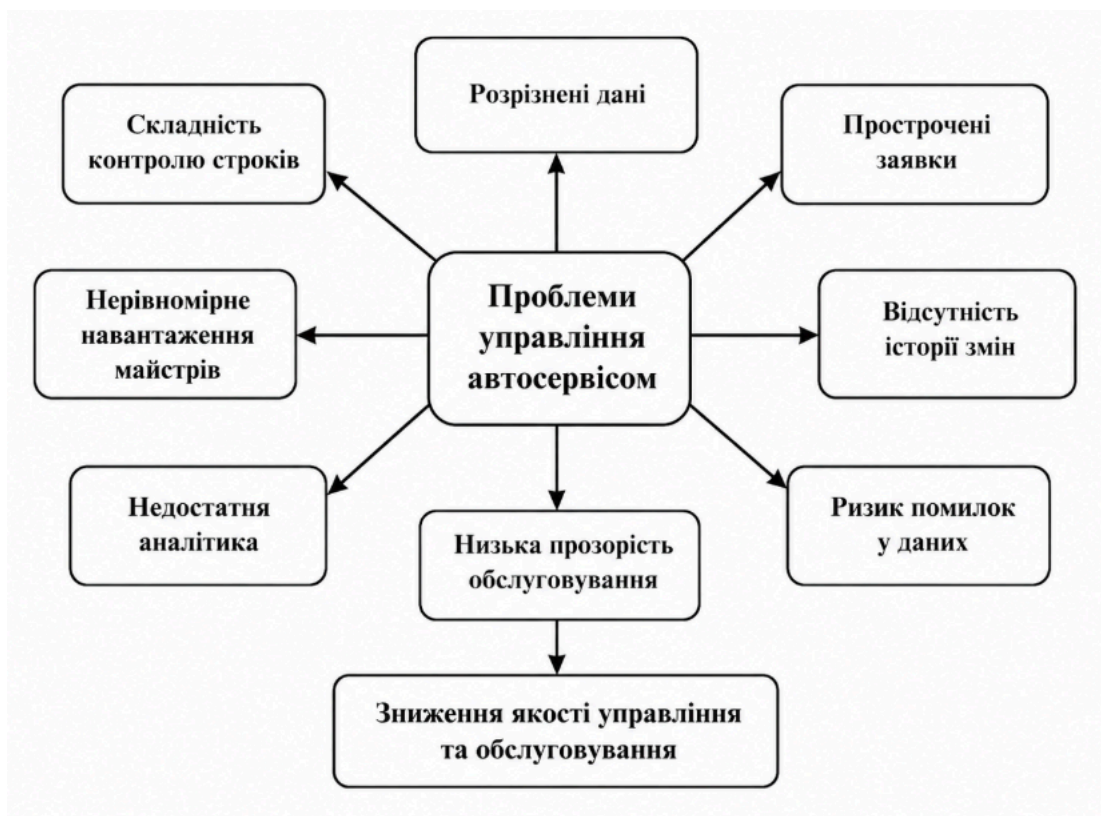


Рис. 1.2 Основні проблеми управління роботою автосервісу

Окремо слід виділити проблему історії змін заявки. У процесі ремонту заявка може змінювати статус, майстра, пріоритет, планову дату завершення або вартість. Якщо ці зміни не фіксуються, складно відновити послідовність подій і зрозуміти, чому виникла затримка або хто саме вносив зміни. Саме тому доцільно передбачити в системі окрему історію подій, яка буде зберігати ключові дії користувачів із заявкою [1; 2; 4; 6].

Ще однією проблемою є обмежений доступ різних ролей до інформації. Адміністратор, менеджер і майстер мають різні завдання, тому повинні бачити різний набір функцій. Адміністратор повинен мати повний доступ до користувачів, фінансових показників, аналітики та довідників. Менеджеру потрібен доступ до клієнтів, автомобілів, заявок і призначення майстрів. Майстер має працювати лише зі своїми заявками та фіксувати виконані роботи. Якщо доступ не розмежований, це може створити ризики помилкового редагування даних або перегляду інформації, яка не потрібна конкретному працівнику [1; 2; 4; 6].

Отже, основними проблемами предметної галузі є розрізнене зберігання інформації, відсутність оперативного контролю строків, складність оцінювання завантаженості майстрів, недостатня аналітика, відсутність історії змін і потреба в рольовому доступі. Вирішення цих проблем можливе шляхом розробки єдиної веб-системи ServiceControl, яка поєднує функції обліку, контролю та аналізу роботи автосервісу [1; 2; 4; 6].

1.3 Аналіз існуючих програмних засобів

Для автоматизації роботи автосервісів можуть використовуватися різні програмні засоби, які забезпечують ведення бази клієнтів, облік транспортних засобів, створення заявок, призначення робіт майстрам, контроль строків виконання, формування рахунків, облік виконаних робіт, збір відгуків і перегляд аналітичних показників [1; 3; 6].

У межах аналізу було розглянуто такі програмні рішення: AutoLear, Shopmonkey, AutoFluent та ARI Auto Repair Software. Вони мають схожу спрямованість, однак відрізняються набором функцій, складністю впровадження, орієнтацією на ринок і зручністю для малих автосервісів [10; 11; 12; 13].

AutoLear є хмарною платформою для керування роботою автосервісу. Система дозволяє створювати робочі замовлення, вести клієнтську базу, організовувати записи, контролювати ремонтні роботи, використовувати цифрові перевірки автомобілів, працювати з відгуками клієнтів і формувати звітність. Перевагою AutoLear є наявність широкого функціоналу та хмарної архітектури, що дозволяє працювати із системою з різних пристроїв. Однак система орієнтована переважно на іноземний ринок, тому її впровадження в українському автосервісі може вимагати адаптації. Крім того, для малого підприємства функціонал може бути надмірним, а вартість використання — завищеною [10].

Shopmonkey є комплексним програмним засобом для управління автосервісом. Він поєднує CRM-функції, облік заявок, планування, виставлення рахунків, роботу з інвентарем, комунікацію з клієнтами та формування звітності.

Система має сучасний інтерфейс і дозволяє керувати багатьма процесами автосервісу в одному середовищі. До переваг Shopmonkey можна віднести підтримку заявок, клієнтів, майстрів, складу та звітів. Недоліком є складність впровадження для невеликих автосервісів, а також орієнтація на англomовний ринок, що може ускладнювати локалізацію та адаптацію бізнес-процесів [11].

AutoFluent є програмним забезпеченням для автосервісів і шинних центрів. Воно містить інструменти для керування клієнтами, замовленнями, інвентарем, працівниками та фінансовими операціями. Особливістю AutoFluent є підтримка відстеження роботи механіків, обліку робочого часу, формування звітів і контролю складських залишків. Такий функціонал є корисним для середніх і великих сервісів, однак для малих автосервісів система може бути складною та вимагати додаткового навчання персоналу [12].

ARI Auto Repair Software є рішенням для обліку клієнтів, автомобілів, рахунків, кошторисів і ремонтних робіт. Водночас її аналітичні можливості є обмеженими, а акцент зроблено переважно на веденні обліку та формуванні документів. Для задачі комплексного аналізу якості обслуговування такого функціоналу недостатньо [13].

Зведене порівняння існуючих програмних засобів і розроблюваної системи ServiceControl наведено в таблиці 1.1.

Таблиця 1.1

Порівняння програмних засобів для автоматизації роботи автосервісу

Показник	AutoLeap	Shopmonkey	AutoFluent	ARI Auto Repair Software	ServiceControl
Основне призначення	Керування автосервісом, заявками та звітністю	Комплексне керування автосервісом	Облік замовлень, персоналу та складу	Облік клієнтів, авто, рахунків і ремонтів	Управління заявками, персоналом, клієнтами та аналітикою стану сервісу

Порівняння програмних засобів для автоматизації роботи автосервісу

Показник	AutoLea p	Shopmonke y	AutoFlue nt	ARI Auto Repair Software	ServiceContr ol
Облік клієнтів	+	+	+	+	+
Облік автомобілів	+	+	+	+	+
Створення заявок	+	+	+	+	+
Призначення майстрів	+	+	+	Без аналізу завантаження	+
Контроль строків виконання	+	+	+	Без оцінки ризиків	+
Облік виконаних робіт	+	+	+	+	+
Робота з відгуками клієнтів	+	Обмежені оцінки	Базові відгуки	Базові відгуки	+
Рольовий доступ	+	+	+	Обмежені ролі	+
Аналітична панель	+	+	+	Мінімальні звіти	+
Оцінка завантаженості майстрів	+	+	+	Без детального аналізу	+
Виявлення проблемних категорій робіт	Через звіти	Через звіти	Через звіти	-	+
Інтегральний індекс стану сервісу	-	-	-	-	+
Історія змін заявки	Журнал дій	Журнал дій	Журнал дій	-	+
Орієнтація на український малий/середній автосервіс	-	-	-	Без локальної адаптації	+

Аналіз існуючих програмних засобів показав, що більшість рішень мають базові функції для автоматизації автосервісу: облік клієнтів, автомобілів, заявок, майстрів і виконаних робіт. Однак вони переважно орієнтовані на комерційне використання, іноземний ринок або великі сервісні підприємства. Крім того, у розглянутих системах недостатньо виражений модуль комплексного аналізу якості обслуговування саме у форматі інтегрального показника стану автосервісу [1; 2; 4; 6].

Саме тому доцільною є розробка системи ServiceControl, яка буде орієнтована на централізований облік процесів автосервісу, розмежування доступу між ролями користувачів і формування аналітичного індексу Service Health Radar. Такий підхід дозволяє не тільки вести заявки, а й швидко виявляти проблемні зони в роботі сервісу [1; 3; 6].

1.4 Обґрунтування доцільності розробки системи ServiceControl

ServiceControl — це веб-система для управління та аналізу роботи автосервісу. Її основна ідея полягає не лише у веденні заявок на ремонт, а й у наданні адміністрації інструментів для контролю стану сервісу. Система дозволяє створювати заявки, вести базу клієнтів і транспортних засобів, призначати майстрів, фіксувати виконані роботи, облікований час, вартість ремонту, відгуки клієнтів та історію змін заявки [1; 2; 4; 6].

На відміну від звичайного електронного журналу заявок, ServiceControl передбачає аналітичний модуль Service Health Radar. Цей модуль оцінює загальний стан автосервісу на основі поточних даних: кількості прострочених заявок, завантаженості майстрів, наявності заявок без виконавця, середньої оцінки клієнтів, тривалості ремонтів і проблемних категорій робіт. У результаті система формує інтегральний показник стану сервісу та список рекомендацій для адміністратора [1; 3; 6].

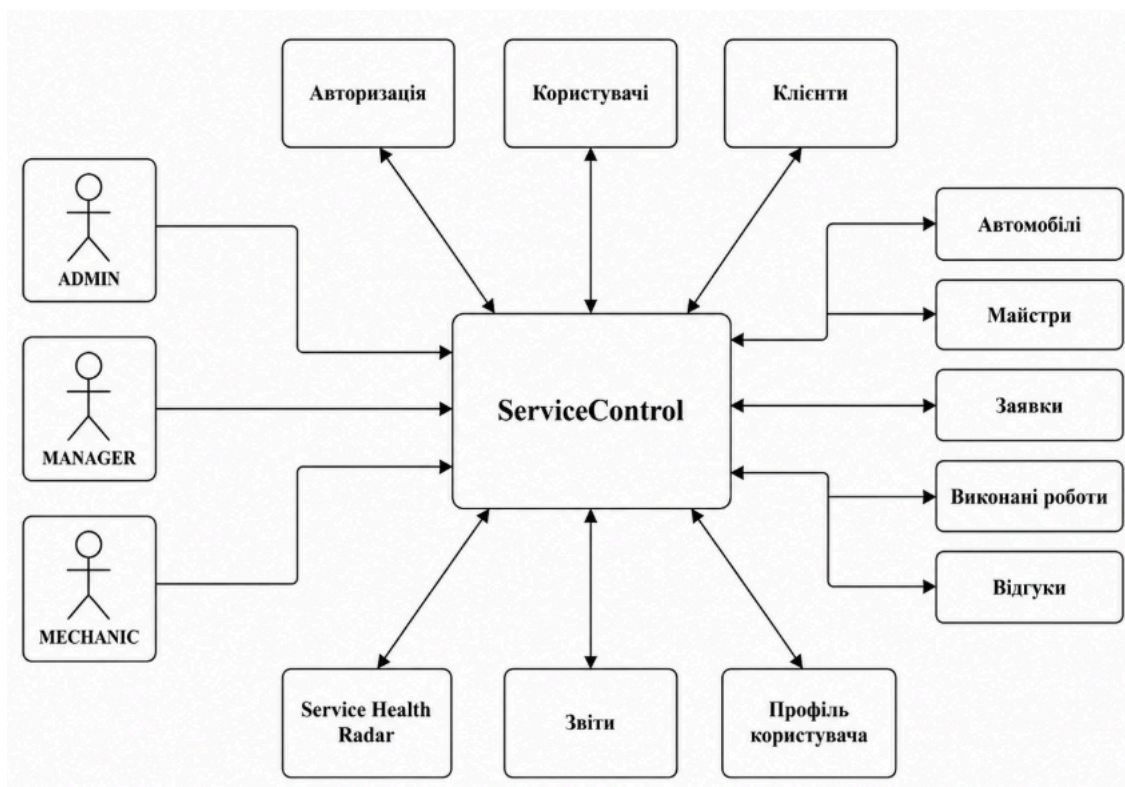


Рис. 1.3 Концептуальна модель системи ServiceControl

У системі передбачено три основні ролі користувачів: ADMIN, MANAGER і MECHANIC. Роль ADMIN призначена для власника або адміністратора автосервісу, який має повний доступ до всіх модулів системи, включаючи управління користувачами, перегляд фінансових показників, аналітики та довідників. Роль MANAGER призначена для працівника, який працює з клієнтами, автомобілями та заявками, призначає майстрів і контролює поточні ремонтні роботи. Роль MECHANIC призначена для майстра, який бачить лише свої призначені заявки, змінює статуси робіт і фіксує виконані дії [1; 2; 4; 6].

Розмежування доступу дозволяє зробити систему зручною для кожної групи користувачів. Адміністратор отримує повну картину стану сервісу, менеджер — інструмент для щоденної роботи із заявками, а майстер — простий інтерфейс для перегляду своїх завдань і внесення результатів виконання робіт. Такий підхід зменшує інформаційне перевантаження та підвищує точність роботи з даними [1; 2; 4; 6].

До основних модулів системи ServiceControl належать:

1. модуль авторизації та реєстрації користувачів;
2. модуль профілю користувача;
3. модуль управління клієнтами;
4. модуль управління автомобілями;
5. модуль управління майстрами;
6. модуль заявок на ремонт;
7. модуль виконаних робіт;
8. модуль відгуків клієнтів;
9. модуль управління користувачами;
10. аналітичний модуль Service Health Radar;
11. модуль звітів.

Центральним елементом системи є заявка на ремонт. Вона пов'язує між собою клієнта, автомобіль, майстра, опис проблеми, статус виконання, пріоритет, планову дату завершення, фактичну дату завершення, виконані роботи, загальну вартість та відгук клієнта. Завдяки цьому вся інформація про процес ремонту зберігається в одному місці, а адміністрація може швидко аналізувати як окрему заявку, так і загальний стан автосервісу [1; 2; 4; 6].

Додатковою перевагою ServiceControl є наявність таблиці подій заявки. Вона дозволяє фіксувати важливі зміни: створення заявки, зміну статусу, призначення майстра, додавання виконаної роботи, завершення ремонту або отримання відгуку. Це підвищує прозорість роботи системи та дозволяє відновити історію дій у разі виникнення спірних ситуацій [1; 2; 4; 6].

Таким чином, розробка ServiceControl є доцільною, оскільки система вирішує одразу декілька задач: централізує дані автосервісу, спрощує роботу персоналу, забезпечує рольовий доступ, дозволяє контролювати строки виконання заявок, аналізувати завантаженість майстрів, враховувати відгуки клієнтів і формувати загальний показник стану сервісу [1; 2; 4; 6].

1.5 Визначення вимог до програмного забезпечення

На основі аналізу предметної галузі, проблем управління роботою автосервісу та існуючих програмних засобів було визначено основні вимоги до системи ServiceControl. Вимоги поділено на функціональні та нефункціональні. Функціональні вимоги описують можливості, які повинна надавати система користувачам, а нефункціональні визначають якісні характеристики її роботи: продуктивність, зручність, безпеку, надійність, підтримуваність і можливість подальшого розвитку [1; 2; 4; 6].

Функціональні вимоги до системи ServiceControl:

- авторизація користувачів за email та паролем;
- реєстрація нових користувачів із зазначенням імені, email, пароля, телефону, посади та ролі;
- підтримка трьох ролей користувачів: ADMIN, MANAGER та MECHANIC;
- розмежування доступу до функцій системи відповідно до ролі користувача;
- перегляд, створення, редагування та деактивація користувачів адміністратором;
- перегляд і редагування профілю поточного користувача;
- ведення бази клієнтів автосервісу;
- додавання, редагування, перегляд і пошук клієнтів за іменем, телефоном або email;
- ведення бази автомобілів клієнтів;
- збереження інформації про марку, модель, номерний знак, VIN-код, рік випуску та власника автомобіля;
- перегляд історії ремонтів конкретного клієнта або автомобіля;
- ведення списку майстрів автосервісу;
- збереження інформації про спеціалізацію майстра, телефон, погодинну ставку, активність і поточне навантаження;
- прив'язка майстра до облікового запису користувача з роллю MECHANIC;
- створення заявки на ремонт автомобіля;

- вибір клієнта, автомобіля, типу ремонту, опису проблеми, пріоритету та планової дати завершення під час створення заявки;
- редагування основних даних заявки;
- призначення відповідального майстра на заявку;
- зміна статусу заявки;
- підтримка статусів NEW, IN_PROGRESS, DONE та CANCELLED;
- підтримка пріоритетів LOW, NORMAL та HIGH;
- пошук і фільтрація заявок за статусом, майстром, клієнтом, номером автомобіля, типом ремонту та пріоритетом;
- перегляд детальної інформації про заявку;
- додавання до заявки виконаних робіт;
- фіксація назви роботи, категорії, витраченого часу, вартості та коментаря майстра;
- облік загальної вартості ремонту;
- додавання відгуку клієнта після завершення заявки;
- збереження оцінки клієнта та текстового коментаря;
- фіксація історії подій заявки;
- збереження інформації про створення заявки, зміну статусу, призначення майстра, додавання роботи, завершення заявки та отримання відгуку;
- відображення головної аналітичної сторінки Dashboard;
- відображення кількості активних, завершених і прострочених заявок;
- відображення середньої оцінки клієнтів;
- відображення завантаженості майстрів;
- визначення заявок без призначеного майстра;
- визначення проблемних категорій ремонтних робіт;
- формування інтегрального показника стану автосервісу Service Health Radar;
- формування рекомендацій для адміністратора на основі аналітичних показників;
- перегляд звітів за заявками, майстрами, відгуками, категоріями робіт і загальними показниками діяльності автосервісу;

- можливість експорту власного звіту системи у форматі PDF.

Нефункціональні вимоги до системи ServiceControl:

- основні операції системи повинні виконуватися протягом 1–2 секунд;
- інтерфейс системи повинен бути зрозумілим для адміністратора, менеджера та майстра;
- сторінки системи повинні мати логічну структуру без надмірного інформаційного перевантаження;
- веб-інтерфейс повинен коректно відображатися на різних розмірах екранів;
- система повинна забезпечувати захищене збереження паролів користувачів;
- доступ до функцій системи повинен надаватися відповідно до ролі користувача;
- заявка повинна бути пов'язана з існуючим клієнтом і автомобілем;
- система повинна коректно обробляти помилки введення даних;
- обов'язкові поля форм повинні перевірятися перед збереженням;
- архітектура системи повинна дозволяти додавати нові модулі в майбутньому;
- програмний код повинен бути розділений на клієнтську частину, серверну частину, сервіси бізнес-логіки та модулі роботи з базою даних;
- основні функції системи повинні бути придатними до ручного та автоматизованого тестування;
- система повинна мати можливість запуску в локальному середовищі для демонстрації та перевірки працездатності;
- аналітичний модуль повинен бути побудований так, щоб у майбутньому можна було додати нові показники;
- показники Dashboard і Service Health Radar повинні бути подані у зрозумілій формі;
- система повинна зменшувати кількість ручної роботи під час ведення заявок і контролю ремонтів;
- структура системи повинна забезпечувати подальшу підтримку, оновлення та розширення функціональності.

2 ЗАСОБИ РЕАЛІЗАЦІЇ СИСТЕМИ SERVICECONTROL

2.1 Обґрунтування вибору клієнт-серверної архітектури

Для реалізації системи ServiceControl було обрано клієнт-серверну архітектуру, оскільки вона дозволяє логічно розділити інтерфейс користувача, бізнес-логіку та збереження даних. Такий підхід є доцільним для веб-системи управління автосервісом, адже різні користувачі системи — адміністратор, менеджер і майстер — повинні працювати з єдиними даними, але через різні функціональні інтерфейси [1; 2; 4; 6].

Клієнт-серверна архітектура передбачає наявність клієнтської частини, серверної частини та бази даних. Клієнтська частина відповідає за відображення інтерфейсу, введення даних користувачем, надсилання запитів до сервера та візуалізацію отриманих результатів. Серверна частина обробляє запити, реалізує бізнес-логіку, перевіряє права доступу, працює з базою даних і формує відповіді для клієнта. База даних забезпечує збереження структурованої інформації про користувачів, клієнтів, автомобілі, майстрів, заявки, виконані роботи, відгуки та історію подій [7; 8; 9].

Для системи ServiceControl такий підхід є особливо важливим, оскільки вона працює з великою кількістю взаємопов'язаних сутностей. Наприклад, заявка на ремонт пов'язана з клієнтом, автомобілем, майстром, виконаними роботами, відгуком і подіями зміни статусу. Якщо всі ці дані зберігати централізовано, система може швидко формувати аналітичні показники, контролювати строки виконання заявок і відображати актуальний стан автосервісу на сторінці Dashboard [1; 3; 6].

Клієнт-серверна архітектура також спрощує подальший розвиток системи. У майбутньому до ServiceControl можна додати мобільний застосунок для майстрів, окремий модуль онлайн-запису клієнтів, інтеграцію з платіжними сервісами або розширену фінансову аналітику. Для цього не потрібно повністю змінювати

структуру системи, оскільки серверна частина вже може надавати дані через API [1; 2; 4; 6].

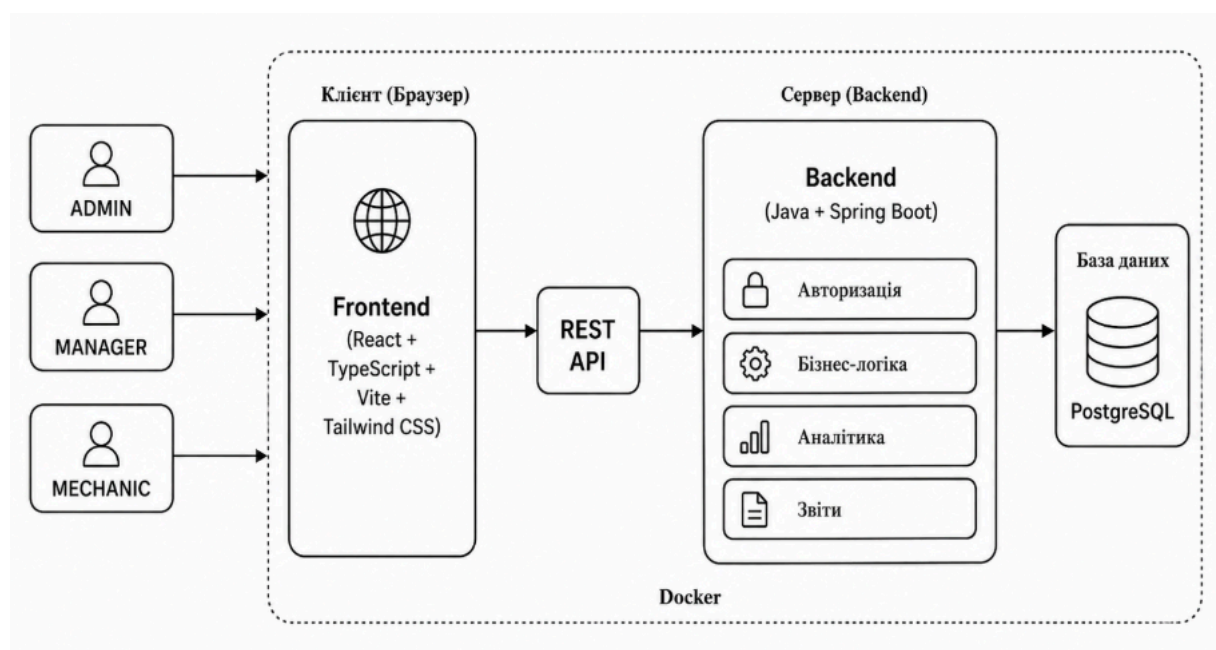


Рис. 2.1 Загальна клієнт-серверна архітектура системи ServiceControl

У межах розробки ServiceControl загальна структура програмної реалізації складається з таких частин:

- Frontend — клієнтська частина системи, створена за допомогою React і TypeScript;
- Backend — серверна частина, реалізована мовою Java з використанням Spring Boot;
- Database — реляційна база даних PostgreSQL;
- REST API — набір запитів для обміну даними між клієнтською та серверною частинами;
- Docker-середовище — засіб для запуску компонентів системи в ізольованому середовищі.

Обраний підхід дозволяє створити структуровану, масштабовану та зручну для супроводу систему. Кожен компонент виконує окрему роль, а взаємодія між ними відбувається через чітко визначені інтерфейси. Це підвищує стабільність програмного забезпечення та спрощує його тестування [7; 8; 9].

2.2 Засоби реалізації серверної частини

Серверна частина системи ServiceControl реалізується мовою програмування Java з використанням фреймворку Spring Boot. Java є однією з найпоширеніших мов для розробки корпоративних веб-систем, оскільки вона забезпечує стабільність, типізацію, надійну роботу з об'єктно-орієнтованими моделями та підтримку великої кількості бібліотек. Для системи управління автосервісом це важливо, оскільки програмне забезпечення повинно працювати з багатьма сутностями, зв'язками, перевірками та ролями користувачів [14; 15].

Spring Boot використовується для швидкого створення серверної частини веб-застосунку. Він спрощує налаштування проєкту, дозволяє створювати REST-контролери, сервіси, репозиторії, підключати базу даних і реалізовувати логіку обробки запитів. У системі ServiceControl серверна частина відповідає за обробку всіх основних операцій: авторизацію користувачів, роботу з клієнтами, автомобілями, майстрами, заявками, виконаними роботами, відгуками, історією подій і аналітичними показниками [14; 15].

Для побудови серверної частини доцільно використовувати декілька основних компонентів Spring-екосистеми. Spring Web забезпечує створення REST API та обробку HTTP-запитів. Spring Data JPA використовується для роботи з базою даних через репозиторії та сутності. Завдяки цьому зменшується кількість повторюваного коду для виконання стандартних операцій створення, читання, оновлення та видалення даних. Spring Security може використовуватися для реалізації авторизації, а також контролю доступу до окремих endpoint-ів залежно від ролі користувача [17].

Серверна частина ServiceControl має багаторівневу структуру. На рівні контролерів приймаються HTTP-запити від клієнтської частини. На рівні сервісів реалізується бізнес-логіка системи: перевірка прав доступу, зміна статусу заявки, розрахунок загальної вартості, формування аналітичних показників, визначення прострочених заявок і завантаженості майстрів. На рівні репозиторіїв виконується

взаємодія з базою даних. Такий підхід дозволяє уникнути змішування різних частин логіки та робить код більш зрозумілим [15; 20].

У системі ServiceControl серверна частина повинна обробляти такі групи запитів:

- авторизація та реєстрація користувачів;
- отримання та редагування профілю користувача;
- управління користувачами;
- управління клієнтами;
- управління автомобілями;
- управління майстрами;
- створення, редагування та перегляд заявок;
- додавання виконаних робіт;
- додавання відгуків клієнтів;
- збереження історії подій заявки;
- формування показників Dashboard;
- розрахунок індексу Service Health Radar;
- формування звітів.

Окреме значення має реалізація аналітичного модуля Service Health Radar. На серверному рівні він повинен отримувати поточні дані із заявок, майстрів, виконаних робіт і відгуків, після чого розраховувати інтегральний показник стану сервісу. До розрахунку можуть входити прострочені заявки, заявки без призначеного майстра, перевантаженість майстрів, низькі оцінки клієнтів, тривалість виконання ремонтів і проблемні категорії робіт [1; 3; 6].

Таким чином, серверна частина є центральним компонентом системи ServiceControl. Вона не лише передає дані між клієнтським інтерфейсом і базою даних, а й реалізує основну бізнес-логіку, контроль доступу та аналітичну обробку інформації [1; 3; 6].

2.3 Засоби реалізації клієнтської частини

Клієнтська частина системи ServiceControl реалізується за допомогою React, TypeScript, Vite та Tailwind CSS. Саме ця частина відповідає за взаємодію користувача із системою, відображення сторінок, форм, списків, аналітичних блоків, таблиць і повідомлень [22].

React було обрано як основний інструмент для створення користувацького інтерфейсу, оскільки він дозволяє будувати веб-застосунки на основі компонентного підходу. Кожна частина інтерфейсу може бути реалізована як окремий компонент: форма входу, форма реєстрації, картка аналітичного показника, таблиця заявок, фільтр, форма створення клієнта, блок Service Health Radar. Такий підхід спрощує повторне використання елементів і підтримку коду [20].

TypeScript використовується для підвищення надійності клієнтської частини. На відміну від звичайного JavaScript, TypeScript дозволяє описувати типи даних, що особливо важливо для роботи з великою кількістю сутностей. Наприклад, для системи ServiceControl можна окремо описати типи User, Client, Vehicle, Mechanic, RepairOrder, RepairWork, Feedback, OrderEvent. Це зменшує кількість помилок під час розробки та робить код більш передбачуваним [21].

Vite використовується як інструмент для створення та запуску frontend-проєкту. Він забезпечує швидкий запуск локального середовища, зручне оновлення сторінок під час розробки та оптимізацію збірки. Це дозволяє швидко перевіряти зміни в інтерфейсі та зручно працювати з React-компонентами [22].

Tailwind CSS застосовується для стилізації інтерфейсу. Він дозволяє швидко створювати адаптивні сторінки, налаштовувати відступи, кольори, сітки, таблиці, кнопки та картки без написання великої кількості окремих CSS-файлів. Для ServiceControl це зручно, оскільки система містить багато однотипних елементів: таблиці, панелі, фільтри, форми, статуси заявок і аналітичні картки [20].

Клієнтська частина ServiceControl включає такі основні сторінки:

- Login — сторінка входу до системи;
- Register — сторінка реєстрації користувача;
- Dashboard — головна аналітична сторінка;
- Repair Orders — сторінка управління заявками на ремонт;
- Order Details — сторінка або панель деталей заявки;
- Clients — сторінка управління клієнтами;
- Vehicles — сторінка управління автомобілями;
- Mechanics — сторінка управління майстрами;
- Users — сторінка управління користувачами для адміністратора;
- Profile — сторінка профілю поточного користувача;
- Reports — сторінка формування звітів.

Особливістю клієнтської частини є те, що інтерфейс повинен змінюватися залежно від ролі користувача. ADMIN повинен бачити повну аналітику, користувачів, довідники, заявки та звіти. MANAGER повинен мати доступ до клієнтів, автомобілів, заявок, майстрів і базової аналітики. MECHANIC повинен бачити тільки свої призначені заявки та мати можливість фіксувати виконані роботи. Такий підхід зменшує перевантаження інтерфейсу та робить систему зручнішою для щоденного використання [1; 2; 4; 6].

Для взаємодії з серверною частиною frontend надсилає HTTP-запити до REST API. Наприклад, під час відкриття сторінки заявок клієнтська частина надсилає запит на отримання списку `repair_orders`, а під час створення нової заявки — запит на збереження нових даних. Після отримання відповіді інтерфейс оновлюється та відображає актуальну інформацію користувачу [15; 20].

Таким чином, клієнтська частина ServiceControl забезпечує зручну взаємодію користувача із системою, відображає дані в зрозумілій формі та дозволяє виконувати основні операції без прямої роботи з базою даних або серверною логікою [1; 2; 4; 6].

2.4 Засоби збереження та обробки даних

Для збереження даних системи ServiceControl використовується реляційна база даних PostgreSQL. Вибір реляційної бази даних є обґрунтованим, оскільки предметна область автосервісу містить багато сутностей, які мають чіткі зв'язки між собою. Користувачі, клієнти, автомобілі, майстри, заявки, виконані роботи, відгуки та події заявки повинні зберігатися структуровано, із забезпеченням цілісності даних [19].

PostgreSQL підходить для реалізації ServiceControl, оскільки підтримує складні зв'язки між таблицями, зовнішні ключі, індекси, транзакції та різні типи даних. Це дозволяє надійно зберігати інформацію про заявки, контролювати зв'язки між сутностями та виконувати аналітичні запити для Dashboard і Service Health Radar [19].

У базі даних ServiceControl передбачено такі основні таблиці:

- users — зберігає дані користувачів системи;
- clients — містить інформацію про клієнтів автосервісу;
- vehicles — зберігає дані про автомобілі клієнтів;
- mechanics — містить інформацію про майстрів;
- repair_orders — зберігає заявки на ремонт;
- repair_works — містить виконані роботи в межах заявки;
- feedback — зберігає оцінки та коментарі клієнтів;
- order_events — фіксує історію змін заявки.

Таблиця users є основою для авторизації та рольового доступу. У ній зберігаються email, пароль у захищеному вигляді, ім'я користувача, роль, телефон, посада, активність акаунта та дати створення або оновлення. На основі ролі користувача система визначає, які сторінки та функції будуть доступні.

Таблиця clients містить дані про клієнтів автосервісу: ім'я, телефон, email, нотатки та дату створення. Таблиця vehicles пов'язана з клієнтами та зберігає інформацію про автомобілі: марку, модель, номерний знак, VIN-код і рік випуску.

Завдяки цьому система може показувати історію ремонтів конкретного клієнта або автомобіля.

Таблиця `mechanics` зберігає дані про майстрів: ім'я, спеціалізацію, телефон, погодинну ставку та активність. Також вона може містити зв'язок з обліковим записом користувача через поле `account_user_id`. Це потрібно для ролі `MECHANIC`, щоб майстер бачив лише власні заявки.

Центральною таблицею є `repair_orders`. Вона містить інформацію про заявку на ремонт: клієнта, автомобіль, майстра, статус, пріоритет, тип сервісу, опис проблеми, дату відкриття, планову дату завершення, фактичну дату завершення та загальну вартість. Саме ця таблиця використовується для більшості управлінських і аналітичних операцій [1; 3; 6].

Таблиця `repair_works` зберігає виконані роботи в межах конкретної заявки. Вона містить назву роботи, категорію, витрачений час, вартість, коментар майстра та дату створення. На основі цих даних можна аналізувати, які категорії робіт виконуються найчастіше, скільки часу вони займають і які з них можуть бути проблемними.

Таблиця `feedback` використовується для збереження оцінки клієнта та коментаря після завершення ремонту. Ці дані важливі для оцінювання якості обслуговування та формування `Service Health Radar`. Якщо середня оцінка клієнтів знижується, система може показати попередження адміністратору.

Таблиця `order_events` призначена для збереження історії змін заявки. У ній можуть фіксуватися події створення заявки, зміни статусу, призначення майстра, додавання виконаних робіт, завершення ремонту або отримання відгуку. Це підвищує прозорість роботи системи та дозволяє аналізувати послідовність дій.

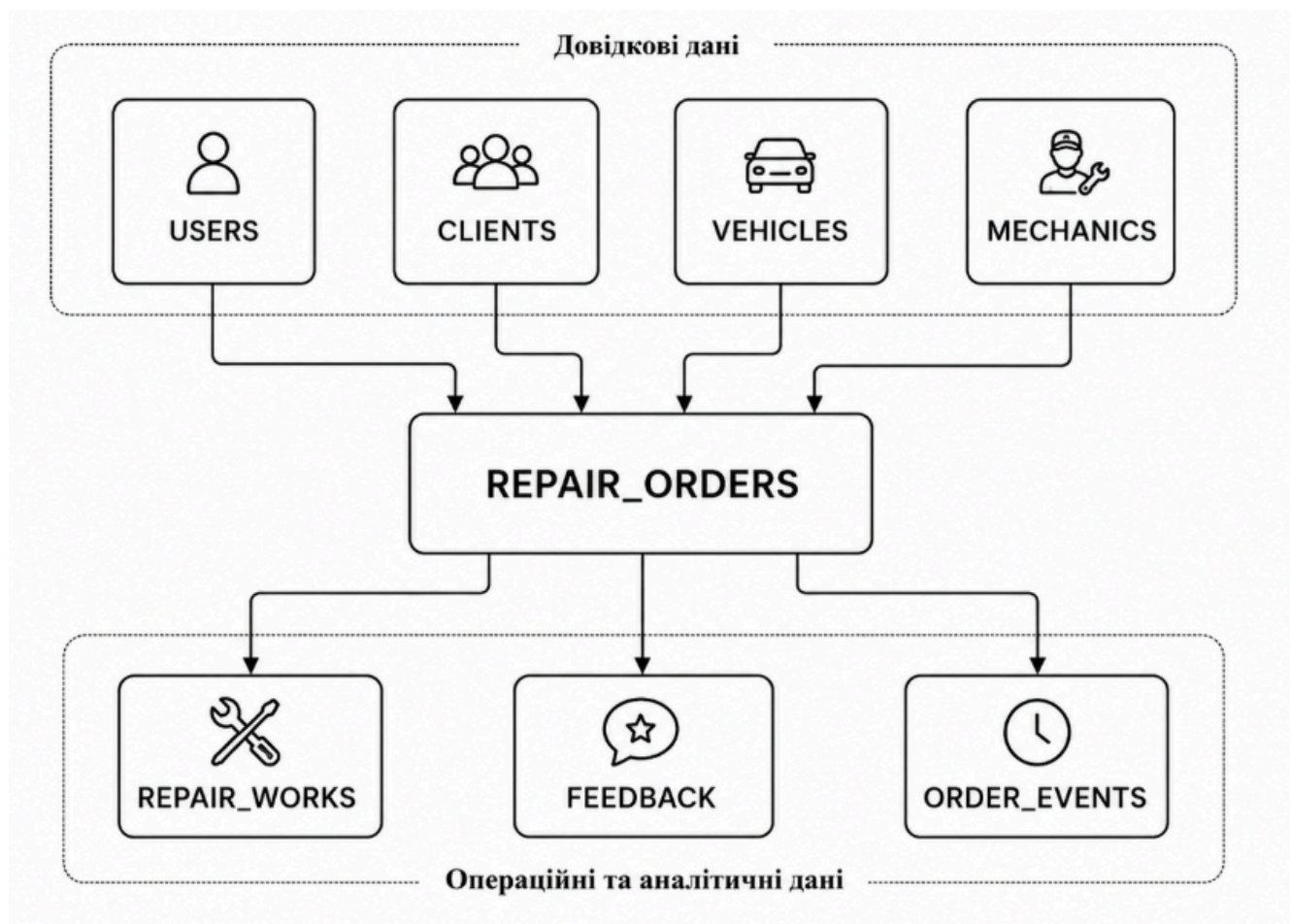


Рис. 2.2 Структура збереження даних у системі ServiceControl

Для взаємодії серверної частини з базою даних використовується Spring Data JPA. Сутності Java відповідають таблицям бази даних, а репозиторії забезпечують виконання операцій з даними. Такий підхід дозволяє працювати з таблицями на рівні об'єктів і зменшує потребу в написанні великої кількості SQL-запитів вручну [17].

Надійне збереження даних є важливим для ServiceControl, оскільки від коректності інформації залежить не тільки облік заявок, а й аналітичні висновки системи. Якщо дані про строки, майстрів, виконані роботи або відгуки будуть неповними, індекс Service Health Radar може бути неточним. Тому структура бази даних повинна забезпечувати повноту, узгодженість і логічну зв'язаність інформації [1; 3; 6].

2.5 Додаткові інструменти розробки та розгортання

Крім основних засобів реалізації, для розробки системи ServiceControl використовуються додаткові інструменти, які спрощують створення, перевірку, запуск і подальшу підтримку програмного забезпечення. До таких інструментів належать Git, GitHub, Docker, середовище розробки, інструменти для тестування API та браузерні засоби розробника [23].

Git використовується як система контролю версій. Він дозволяє зберігати історію змін у проєкті, відстежувати правки в коді, працювати з різними гілками розробки та повертатися до попередніх версій у разі помилок. Для кваліфікаційної роботи це важливо, оскільки проєкт поступово розширюється: спочатку реалізуються базові сутності, потім додаються ролі, аналітика, звіти та тестування [7; 8; 9].

GitHub може використовуватися як віддалене сховище коду. Він дозволяє зберігати проєкт у хмарному репозиторії, демонструвати структуру розробки, переглядати історію комітів і зручно передавати проєкт для перевірки. Також GitHub може містити README-файл з описом системи, інструкцією запуску та коротким описом основних модулів [7; 8; 9].

Docker використовується для контейнеризації компонентів системи. За його допомогою можна запускати frontend, backend і базу даних в ізольованих середовищах. Це спрощує розгортання системи, оскільки всі необхідні компоненти можуть бути налаштовані в одному docker-compose файлі. Наприклад, окремий контейнер може відповідати за PostgreSQL, інший — за Spring Boot backend, а ще один — за React frontend. У звіті практики також зазначено, що Docker дозволяє запускати серверну частину, базу даних і клієнтську частину як окремі контейнери в однаковому середовищі [23].

Для перевірки REST API можуть використовуватися Postman або Swagger. Postman дозволяє вручну надсилати запити до backend-частини, перевіряти

відповіді сервера, тестувати створення заявок, авторизацію, отримання списку клієнтів або формування аналітичних показників. Swagger може бути використаний для документування API та зручного перегляду доступних endpoint-ів [15; 20].

Браузерні інструменти розробника використовуються для перевірки роботи клієнтської частини. За їх допомогою можна аналізувати HTTP-запити, перевіряти відповіді сервера, переглядати помилки JavaScript, тестувати адаптивність сторінок і контролювати стан інтерфейсу [15; 20].

Для написання та підтримки коду можуть використовуватися середовища розробки IntelliJ IDEA та Visual Studio Code. IntelliJ IDEA зручна для роботи з Java та Spring Boot, оскільки підтримує навігацію по класах, автодоповнення, запуск серверної частини та роботу з базою даних. Visual Studio Code зручний для frontend-розробки на React і TypeScript, оскільки має велику кількість розширень для JavaScript, TypeScript, Tailwind CSS і роботи з Git [21].

Усі ці інструменти разом забезпечують повний цикл розробки системи ServiceControl: написання коду, збереження версій, перевірку API, запуск компонентів, тестування інтерфейсу та підготовку проєкту до демонстрації [7; 8; 9].

3 ПРОЄКТУВАННЯ СИСТЕМИ SERVICECONTROL

3.1 Загальна логічна структура системи

Проєктування системи ServiceControl виконувалося з урахуванням основної мети програмного забезпечення — забезпечити централізоване управління роботою автосервісу, контроль заявок на ремонт, ведення бази клієнтів і автомобілів, облік роботи майстрів, фіксацію відгуків клієнтів та формування аналітичних показників стану сервісу [1; 3; 6].

Система побудована за модульним принципом. Це означає, що кожна функціональна частина відповідає за окремий напрям роботи автосервісу. Такий підхід спрощує розробку, тестування, підтримку та подальше розширення системи. Наприклад, модуль заявок відповідає за створення та зміну ремонтних заявок, модуль клієнтів — за збереження контактної інформації, модуль майстрів — за облік персоналу, а аналітичний модуль — за оцінювання загального стану автосервісу [7; 8; 9].

Логічна структура системи ServiceControl включає такі основні модулі:

- модуль автентифікації та авторизації;
- модуль управління користувачами;
- модуль профілю користувача;
- модуль клієнтів;
- модуль автомобілів;
- модуль майстрів;
- модуль заявок на ремонт;
- модуль виконаних робіт;
- модуль відгуків клієнтів;
- модуль історії подій заявки;
- аналітичний модуль Dashboard;
- модуль Service Health Radar;

– модуль звітів.

Модуль автентифікації та авторизації відповідає за вхід користувачів до системи, перевірку email і пароля, а також визначення ролі користувача. Після входу система надає доступ лише до тих сторінок і функцій, які відповідають ролі поточного користувача [7; 8; 9].

Модуль управління користувачами призначений для адміністратора. Він дозволяє переглядати список облікових записів, створювати нових користувачів, змінювати їхні ролі, редагувати базові дані та деактивувати акаунти. Цей модуль потрібен для контролю доступу до системи та підтримки актуального складу працівників автосервісу [1; 2; 4; 6].

Модуль клієнтів забезпечує збереження інформації про клієнтів автосервісу: імені, телефону, електронної пошти та додаткових нотаток. Він дозволяє швидко знайти клієнта, переглянути його автомобілі та історію звернень [1; 2; 4; 6].

Модуль автомобілів призначений для збереження даних про транспортні засоби клієнтів. Для кожного автомобіля фіксуються марка, модель, номерний знак, VIN-код, рік випуску та власник. Це дозволяє пов'язати кожну заявку з конкретним автомобілем і в майбутньому переглядати історію ремонтів [1; 2; 4; 6].

Модуль майстрів використовується для ведення списку працівників, які виконують ремонтні роботи. У ньому зберігається інформація про спеціалізацію майстра, телефон, погодинну ставку, активність та зв'язок з обліковим записом користувача. Завдяки цьому майстер із роллю MECHANIC може бачити лише власні призначені заявки [1; 2; 4; 6].

Модуль заявок на ремонт є центральним модулем системи. Саме в ньому створюються, редагуються та контролюються заявки. Заявка поєднує між собою клієнта, автомобіль, майстра, статус, пріоритет, опис проблеми, строки виконання, виконані роботи, вартість і відгук клієнта [1; 2; 4; 6].

Модуль виконаних робіт дозволяє фіксувати конкретні дії, які були виконані в межах заявки. Для кожної роботи зберігаються назва, категорія, витрачений час, вартість і коментар майстра. Ці дані використовуються не тільки для обліку ремонту, а й для подальшої аналітики [1; 2; 4; 6].

Модуль відгуків клієнтів дозволяє зберігати оцінку та коментар після завершення ремонту. Відгуки є важливим джерелом даних для оцінювання якості обслуговування та формування аналітичних показників [1; 3; 6].

Модуль історії подій заявки фіксує ключові зміни, які відбуваються із заявкою: створення, зміну статусу, призначення майстра, додавання виконаних робіт, завершення ремонту або отримання відгуку. Це підвищує прозорість роботи системи та дозволяє відновити послідовність дій у разі потреби [1; 2; 4; 6].

Аналітичний модуль Dashboard забезпечує відображення основних показників роботи автосервісу. До них належать кількість активних заявок, завершених заявок, прострочених заявок, середня оцінка клієнтів, завантаженість майстрів, фінансові показники та проблемні категорії робіт [1; 3; 6].

Окреме місце займає модуль Service Health Radar. Він оцінює загальний стан автосервісу на основі кількох показників і формує інтегральний індекс. Такий підхід дозволяє адміністратору швидко зрозуміти, чи є в роботі сервісу ризики: перевантажені майстри, прострочені заявки, низькі оцінки клієнтів або часті затримки в певних категоріях ремонту [1; 3; 6].

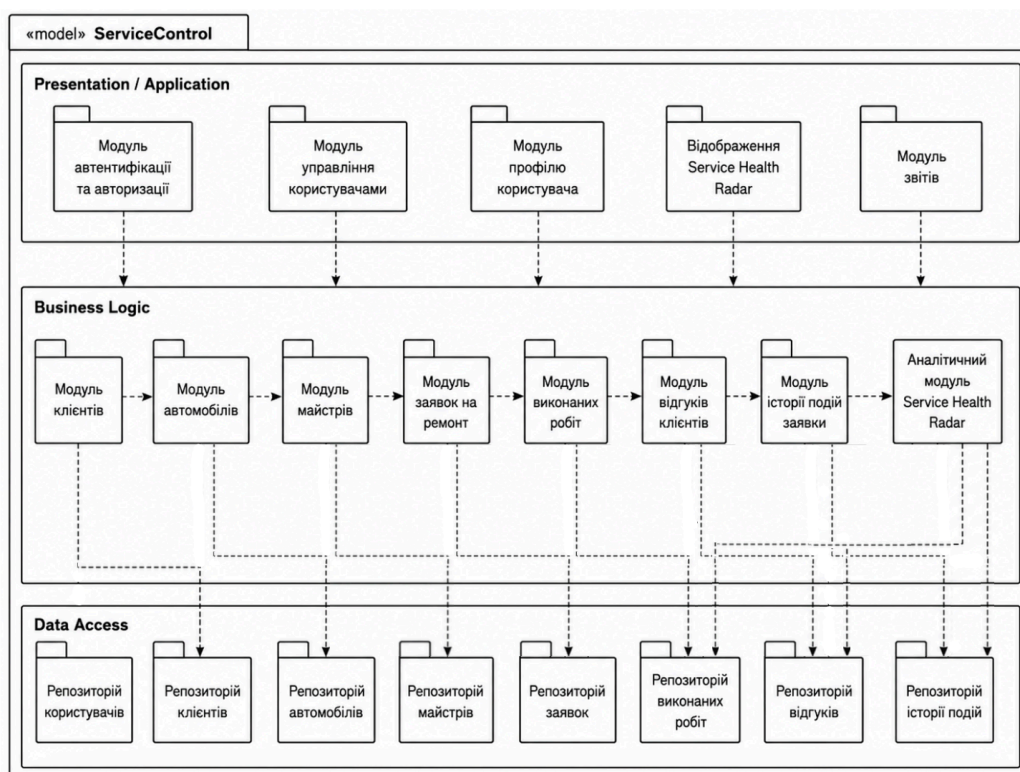


Рис. 3.1 Логічна структура системи ServiceControl

Таким чином, логічна структура ServiceControl побудована навколо заявки на ремонт як основної сутності. Інші модулі доповнюють її, забезпечуючи повний цикл роботи автосервісу: від створення заявки до завершення ремонту, отримання відгуку та аналізу результатів [1; 2; 4; 6].

3.2 Проєктування ролей користувачів та прав доступу

У системі ServiceControl передбачено три основні ролі користувачів: ADMIN, MANAGER та MECHANIC. Такий розподіл відповідає типовій організаційній структурі автосервісу, де різні працівники виконують різні задачі та повинні мати різний рівень доступу до інформації [1; 2; 4; 6].

Роль ADMIN призначена для власника або головного адміністратора автосервісу. Користувач з цією роллю має повний доступ до всіх модулів системи. Він може керувати користувачами, клієнтами, автомобілями, майстрами, заявками, виконаними роботами, відгуками, аналітикою та звітами. Також адміністратор має доступ до повного Dashboard і Service Health Radar [1; 3; 6].

Роль MANAGER призначена для працівника, який працює з клієнтами та організовує процес ремонту. Менеджер може створювати заявки, додавати клієнтів і автомобілі, призначати майстрів, змінювати статуси заявок, контролювати строки виконання та переглядати базову аналітику. Водночас менеджер не повинен мати доступу до управління користувачами та повної адміністративної інформації [1; 2; 4; 6].

Роль MECHANIC призначена для майстра автосервісу. Користувач із цією роллю бачить лише ті заявки, які призначені саме йому. Майстер може переглядати деталі своїх заявок, додавати виконані роботи, вказувати витрачений час, вартість, категорію роботи та коментар. Такий обмежений доступ робить інтерфейс простішим і зменшує ризик випадкового редагування чужих даних [1; 2; 4; 6].

Розмежування доступу є важливим для безпеки та зручності роботи системи. Кожен користувач бачить тільки ті сторінки й функції, які потрібні для виконання

його посадових обов'язків. Це дозволяє уникнути інформаційного перевантаження та зменшує ймовірність помилок.

Права доступу в системі можна описати таким чином:

- ADMIN має повний доступ до всіх сторінок і функцій системи;
- MANAGER має доступ до клієнтів, автомобілів, заявок, майстрів, відгуків і базової аналітики;
- MECHANIC має доступ лише до власних заявок і виконаних робіт;
- тільки ADMIN може створювати, редагувати та деактивувати користувачів;
- тільки ADMIN може змінювати ролі користувачів;
- ADMIN і MANAGER можуть створювати заявки;
- ADMIN і MANAGER можуть призначати майстрів;
- MECHANIC може додавати виконані роботи лише до своїх заявок;
- MECHANIC не має доступу до сторінки користувачів;
- MECHANIC не має доступу до повної фінансової та адміністративної аналітики.

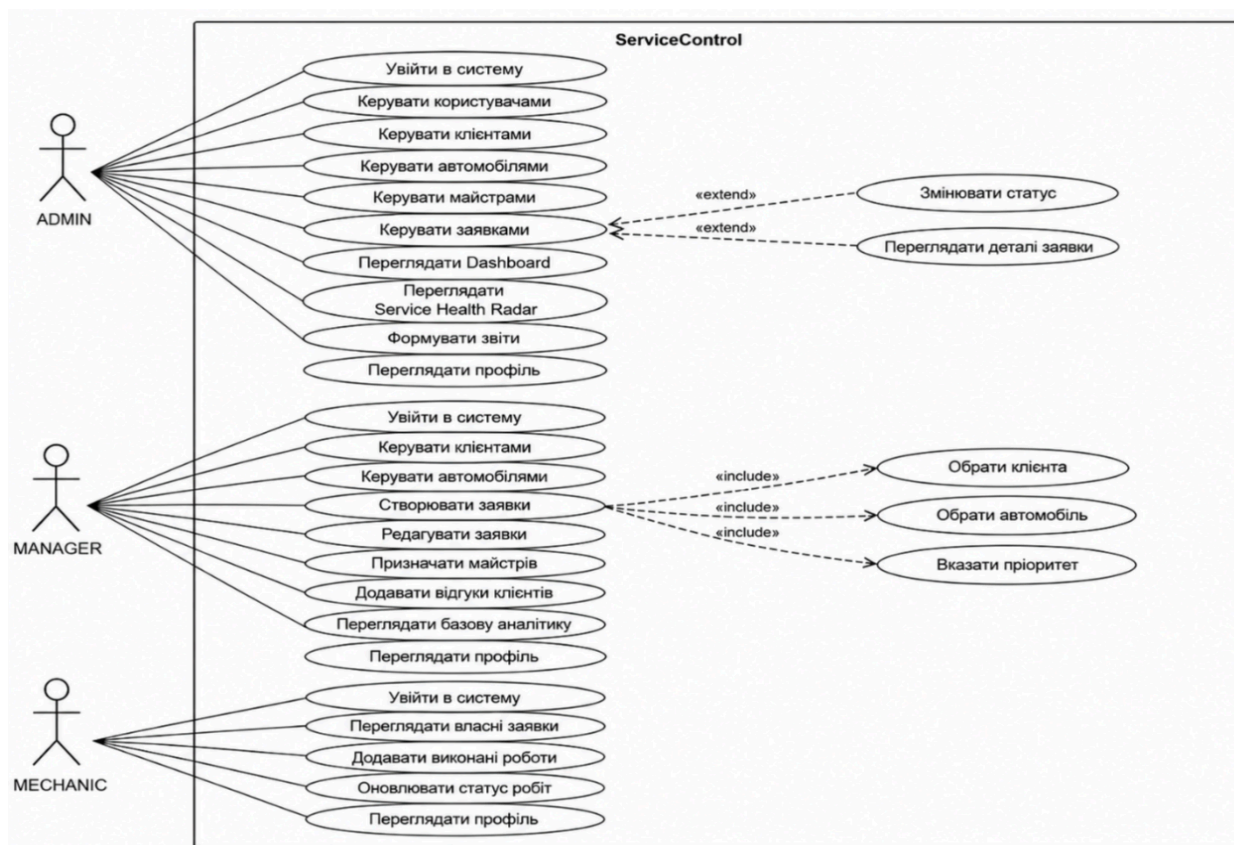


Рис. 3.2 Діаграма варіантів використання системи ServiceControl

Діаграма варіантів використання повинна відобразити взаємодію трьох ролей із системою.

Для ADMIN доцільно показати варіанти використання: «Увійти в систему», «Керувати користувачами», «Керувати клієнтами», «Керувати автомобілями», «Керувати майстрами», «Керувати заявками», «Переглядати аналітику», «Формувати звіти».

Для MANAGER — «Працювати з клієнтами», «Створювати заявки», «Призначати майстрів», «Контролювати статуси заявок», «Додавати відгуки».

Для MECHANIC — «Переглядати власні заявки», «Додавати виконані роботи», «Змінювати статус виконання робіт» [1; 2; 4; 6].

Запропонована рольова модель дозволяє адаптувати систему під реальні процеси автосервісу. Вона забезпечує баланс між функціональністю, безпекою та простотою використання [1; 2; 4; 6].

3.3 Проєктування бази даних

База даних системи ServiceControl повинна забезпечувати збереження всієї інформації, необхідної для роботи автосервісу. Оскільки предметна область містить багато пов'язаних сутностей, для збереження даних використовується реляційна модель. Основними сутностями є користувачі, клієнти, автомобілі, майстри, заявки на ремонт, виконані роботи, відгуки та події заявки [1; 2; 4; 6].

Центральною сутністю бази даних є заявка на ремонт. Вона пов'язує клієнта, автомобіль, майстра, виконані роботи, відгук і події. Через це більшість таблиць або прямо, або опосередковано пов'язані із заявкою [1; 2; 4; 6].

Основні таблиці бази даних ServiceControl:

- USERS;
- CLIENTS;
- VEHICLES;
- MECHANICS;

- REPAIR_ORDERS;
- REPAIR_WORKS;
- FEEDBACK;
- ORDER_EVENTS.

Таблиця USERS призначена для збереження інформації про користувачів системи. Вона містить email, пароль у захищеному вигляді, ім'я користувача, роль, телефон, посаду, активність акаунта та дати створення й оновлення. Поле role визначає, які функції будуть доступні користувачу.

Таблиця CLIENTS містить інформацію про клієнтів автосервісу. До неї входять ім'я клієнта, телефон, email, нотатки та дата створення запису. Клієнт може мати один або декілька автомобілів, а також одну або декілька заявок на ремонт.

Таблиця VEHICLES призначена для збереження інформації про автомобілі. Кожен автомобіль пов'язаний із конкретним клієнтом. У таблиці зберігаються марка, модель, номерний знак, VIN-код і рік випуску. Завдяки цьому система може формувати історію ремонтів для конкретного транспортного засобу.

Таблиця MECHANICS містить дані про майстрів автосервісу. Вона включає ім'я майстра, спеціалізацію, телефон, погодинну ставку, активність і зв'язок з обліковим записом користувача. Поле account_user_id дозволяє пов'язати майстра з користувачем системи, що важливо для ролі MECHANIC.

Таблиця REPAIR_ORDERS є центральною таблицею системи. Вона містить інформацію про заявку на ремонт: клієнта, автомобіль, майстра, статус, пріоритет, тип ремонту, опис проблеми, дату відкриття, планову дату завершення, фактичну дату завершення та загальну вартість.

Таблиця REPAIR_WORKS містить виконані роботи, пов'язані із заявкою. Для кожної роботи зберігаються назва, категорія, витрачений час, вартість, коментар майстра та дата створення. Ці дані використовуються для деталізації заявки та формування аналітики за категоріями робіт.

Таблиця FEEDBACK зберігає відгуки клієнтів. Вона містить оцінку, текстовий коментар, дату створення та зв'язок із заявкою. Відгуки використовуються для оцінювання якості обслуговування та впливають на показники Service Health Radar.

Таблиця ORDER_EVENTS призначена для фіксації історії подій заявки. У ній зберігаються тип події, опис, користувач, який виконав дію, заявка та дата створення події. Завдяки цьому можна простежити, як змінювалася заявка протягом усього циклу обслуговування.

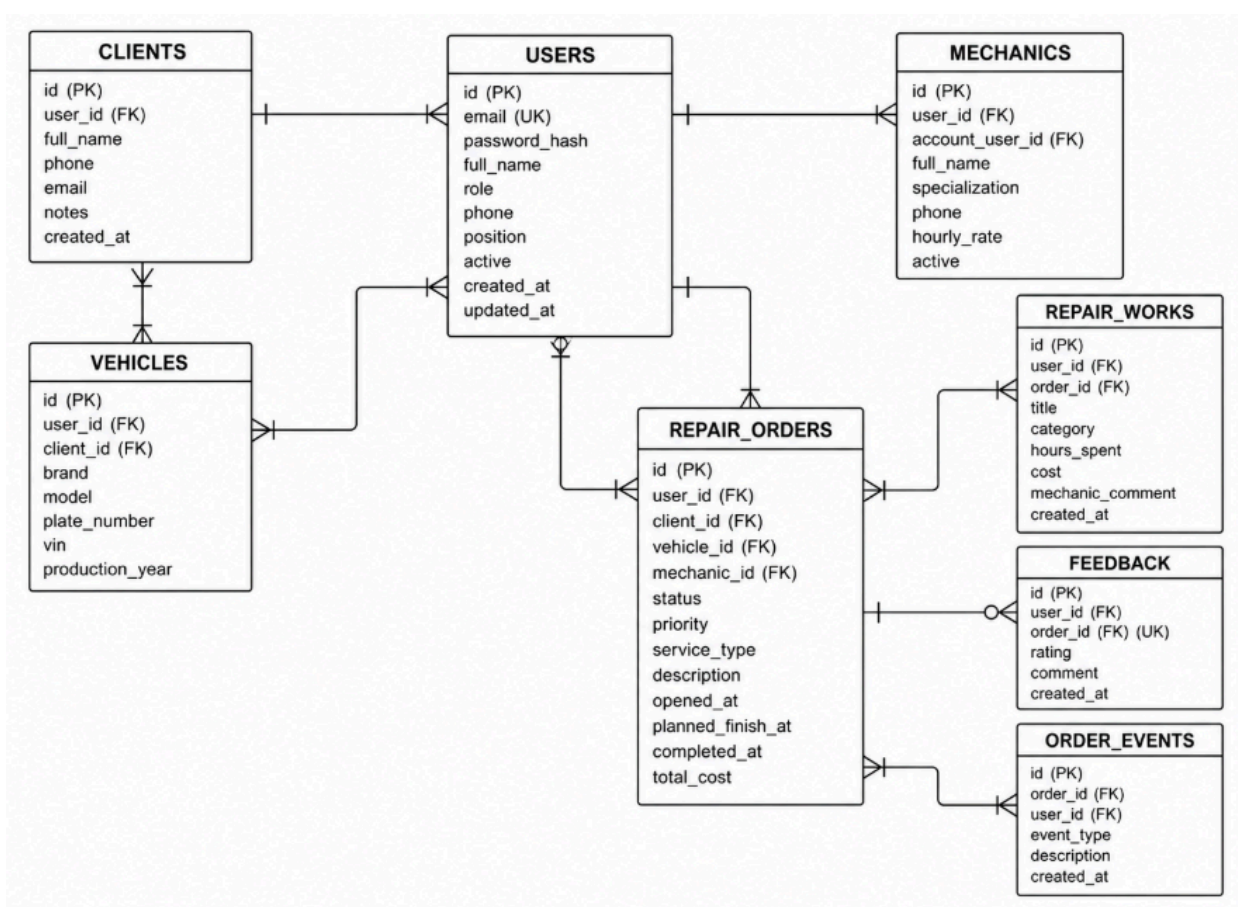


Рис. 3.3 ER-діаграма бази даних системи ServiceControl

Розроблена структура бази даних дозволяє зберігати не тільки поточний стан автосервісу, а й накопичувати дані для подальшого аналізу. Це є необхідною умовою для роботи Dashboard і Service Health Radar [1; 3; 6].

3.4 Проєктування процесу роботи із заявкою на ремонт

Заявка на ремонт є основною сутністю системи ServiceControl. Саме через заявку відбувається зв'язок між клієнтом, автомобілем, майстром, виконаними роботами, строками, вартістю та відгуком. Тому процес роботи із заявкою повинен бути спроектований максимально логічно та послідовно [1; 2; 4; 6].

Життєвий цикл заявки починається зі створення. Менеджер або адміністратор обирає клієнта, автомобіль, вказує тип ремонту, опис проблеми, пріоритет і планову дату завершення. Якщо відповідний клієнт або автомобіль ще не внесені до системи, їх можна додати перед створенням заявки [1; 2; 4; 6].

Після створення заявка отримує статус NEW. На цьому етапі вона ще може не мати призначеного майстра. Якщо майстер не призначений, така заявка повинна відображатися в системі як заявка, що потребує уваги менеджера. Це важливо для аналітичного модуля, оскільки велика кількість заявок без виконавця може свідчити про проблеми в організації роботи [1; 3; 6].

Далі менеджер або адміністратор призначає відповідального майстра. Після початку виконання ремонту статус заявки змінюється на IN_PROGRESS. Майстер бачить цю заявку у своєму інтерфейсі та може додавати виконані роботи. Для кожної роботи вказується назва, категорія, витрачений час, вартість і коментар [1; 2; 4; 6].

Після завершення всіх робіт заявка переводиться у статус DONE. У ній фіксується фактична дата завершення та загальна вартість ремонту. Після цього може бути доданий відгук клієнта: оцінка та коментар. Якщо ремонт було скасовано, заявка може отримати статус CANCELLED [1; 2; 4; 6].

Статуси заявки в системі:

- NEW — нова заявка;
- IN_PROGRESS — заявка перебуває в роботі;
- DONE — заявка завершена;
- CANCELLED — заявка скасована.

Пріоритети заявки:

- LOW — низький пріоритет;
- NORMAL — звичайний пріоритет;
- HIGH — високий пріоритет.

Процес роботи із заявкою можна подати такою послідовністю:

- створення заявки менеджером або адміністратором;
- вибір клієнта та автомобіля;
- введення опису проблеми, типу ремонту, пріоритету та планової дати завершення;
- збереження заявки зі статусом NEW;
- призначення майстра;
- зміна статусу на IN_PROGRESS;
- додавання виконаних робіт майстром;
- розрахунок або фіксація загальної вартості;
- завершення заявки зі статусом DONE;
- додавання відгуку клієнта;
- збереження подій заявки в історії.

У процесі роботи із заявкою система повинна фіксувати ключові події. Наприклад, при створенні заявки створюється подія типу ORDER_CREATED, при призначенні майстра — MECHANIC_ASSIGNED, при зміні статусу — STATUS_CHANGED, при додаванні роботи — WORK_ADDED, при завершенні заявки — ORDER_COMPLETED, при додаванні відгуку — FEEDBACK_ADDED [1; 2; 4; 6].

Фіксація подій має декілька переваг. По-перше, вона дозволяє відстежити історію конкретної заявки. По-друге, вона підвищує прозорість роботи персоналу. По-третє, такі дані можуть використовуватися для аналізу типових затримок або частоти змін статусів [1; 2; 4; 6].

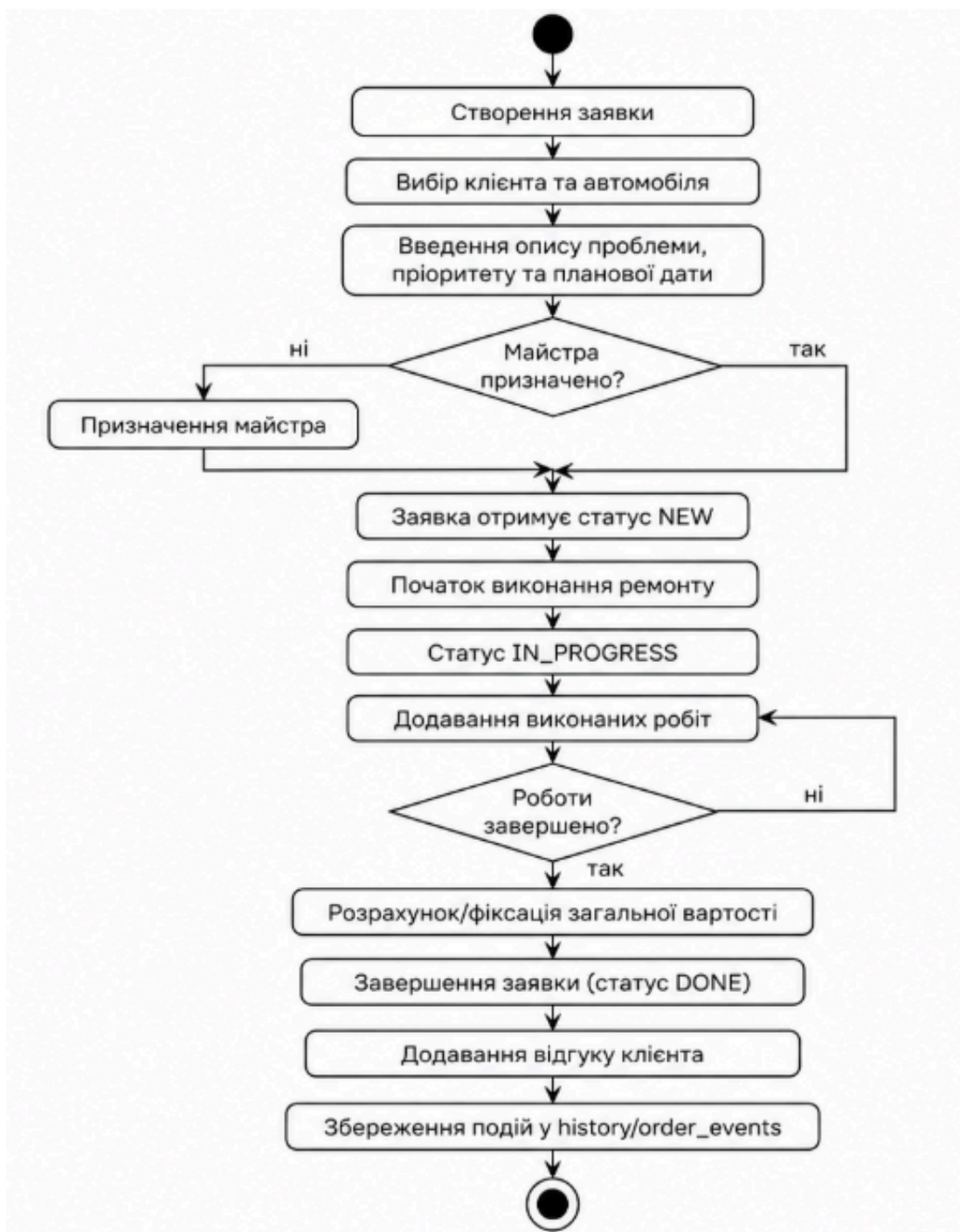


Рис. 3.4 Блок-схема процесу обробки заявки на ремонт

Таким чином, правильно спроектований процес роботи із заявкою дозволяє не тільки автоматизувати облік ремонтів, а й створити основу для аналітики якості обслуговування [1; 2; 4; 6].

3.5 Проектування аналітичного модуля Service Health Radar

Однією з ключових особливостей системи ServiceControl є аналітичний модуль Service Health Radar. Його призначення полягає в оцінюванні загального стану автосервісу на основі поточних даних системи. На відміну від звичайної статистики, Service Health Radar повинен не просто показувати окремі цифри, а формувати узагальнений індекс стану сервісу та виявляти проблемні зони [1; 3; 6].

Service Health Radar може враховувати такі показники:

- кількість прострочених заявок;
- кількість заявок без призначеного майстра;
- завантаженість майстрів;
- середню оцінку клієнтів;
- кількість негативних відгуків;
- середню тривалість ремонту;
- проблемні категорії виконаних робіт;
- кількість активних заявок;
- співвідношення завершених і незавершених заявок.

Прострочені заявки є одним із найважливіших показників. Якщо заявка має планову дату завершення, але не була виконана вчасно, вона повинна впливати на зниження загального індексу стану сервісу. Велика кількість прострочених заявок може свідчити про перевантаження персоналу, неправильне планування або проблеми з організацією ремонту [1; 2; 4; 6].

Заявки без призначеного майстра також є проблемним показником. Якщо заявка створена, але не має виконавця, це означає, що процес ремонту фактично ще не організований. Такі заявки повинні відображатися в Dashboard і впливати на рекомендації для менеджера або адміністратора [1; 3; 6].

Завантаженість майстрів дозволяє оцінити, наскільки рівномірно розподілені заявки між працівниками. Якщо один майстер має значно більше активних заявок,

ніж інші, система може позначити його як перевантаженого. Це допомагає адміністрації вчасно перерозподілити роботу [1; 2; 4; 6].

Середня оцінка клієнтів використовується для аналізу якості обслуговування. Якщо оцінка знижується, це може вказувати на проблеми в комунікації, затримки ремонту, невідповідність вартості очікуванням клієнта або інші фактори. Низькі відгуки повинні впливати на загальний індекс Service Health [1; 2; 4; 6].

Проблемні категорії робіт визначаються на основі виконаних ремонтних робіт. Якщо певна категорія, наприклад «Підвіска» або «Двигун», часто пов'язана із затримками, високою вартістю або низькими оцінками клієнтів, система повинна виводити її як ризикову категорію [1; 2; 4; 6].

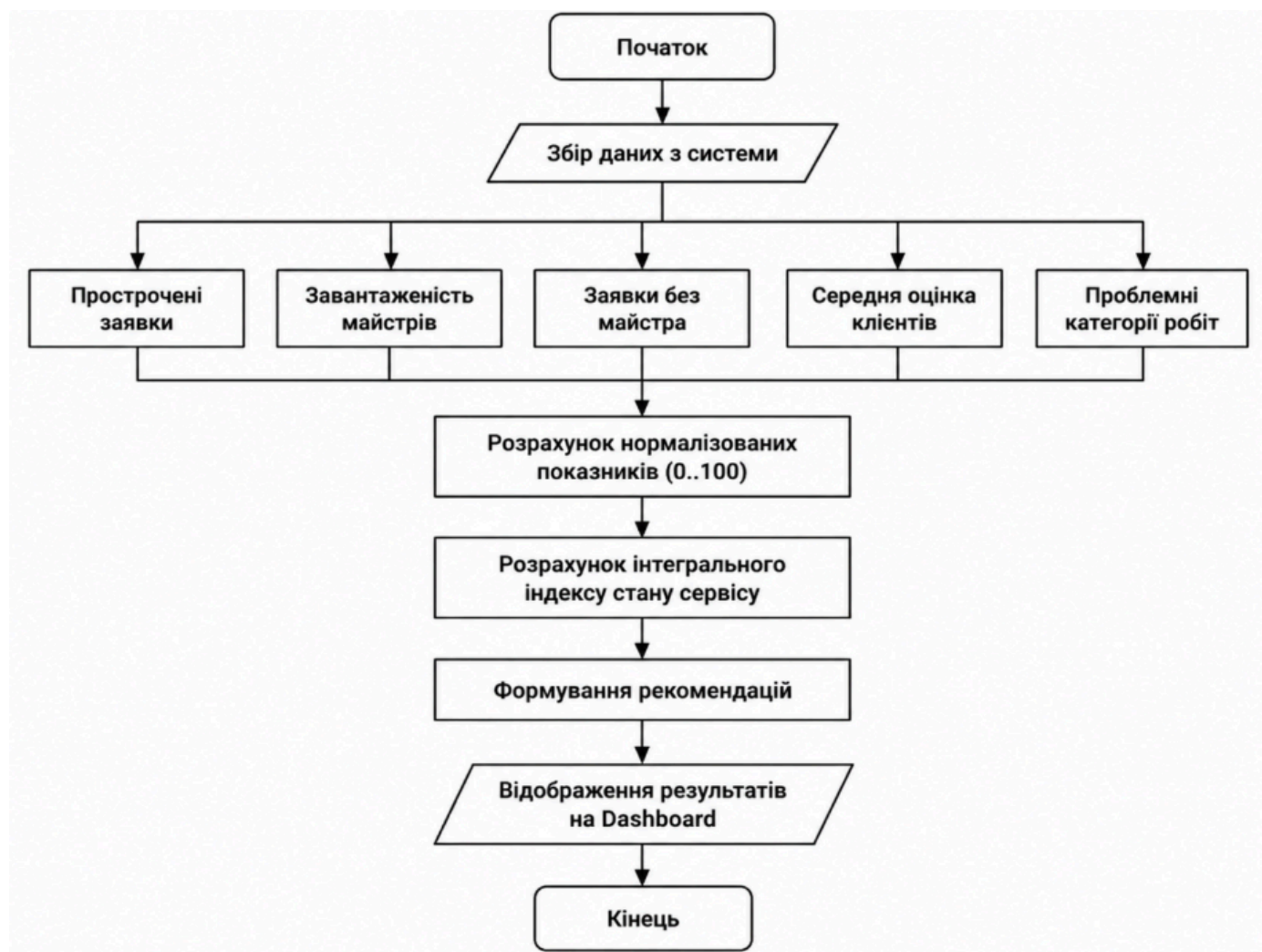


Рис. 3.5 Логіка формування індексу Service Health Radar

Індекс Service Health може подаватися у вигляді значення від 0 до 100. Чим вищий показник, тим стабільнішим є стан автосервісу. Наприклад, значення 80–100 може означати нормальний стан сервісу, 60–79 — наявність окремих ризиків, нижче 60 — потребу в управлінському втручанні [1; 2; 4; 6].

Такий модуль має практичну цінність, оскільки допомагає адміністрації не просто переглядати заявки, а швидко розуміти, де саме виникають проблеми. Це перетворює ServiceControl з облікової системи на інструмент підтримки управлінських рішень [1; 2; 4; 6].

3.6 Проєктування API та взаємодії компонентів системи

Взаємодія між клієнтською та серверною частинами системи ServiceControl здійснюється через REST API. Такий підхід дозволяє відокремити інтерфейс користувача від бізнес-логіки та забезпечити чіткий обмін даними між frontend і backend [15; 20].

Клієнтська частина надсилає HTTP-запити до серверної частини, а сервер обробляє їх, взаємодіє з базою даних і повертає результат у форматі JSON. Наприклад, при відкритті сторінки заявок frontend надсилає запит на отримання списку заявок. Backend отримує дані з бази, формує відповідь і повертає її клієнтській частині. Після цього React-компоненти відображають актуальні дані користувачу [20].

У системі ServiceControl API можна умовно поділити на такі групи:

- /auth — авторизація та реєстрація користувачів;
- /users — управління користувачами;
- /profile — робота з профілем поточного користувача;
- /clients — робота з клієнтами;
- /vehicles — робота з автомобілями;
- /mechanics — робота з майстрами;
- /repair-orders — створення, редагування та перегляд заявок;

- /repair-works — робота з виконаними ремонтними роботами;
- /feedback — додавання та перегляд відгуків клієнтів;
- /order-events — перегляд історії подій заявки;
- /dashboard — отримання основних аналітичних показників;
- /service-health — отримання індексу Service Health Radar;
- /reports — формування звітів.

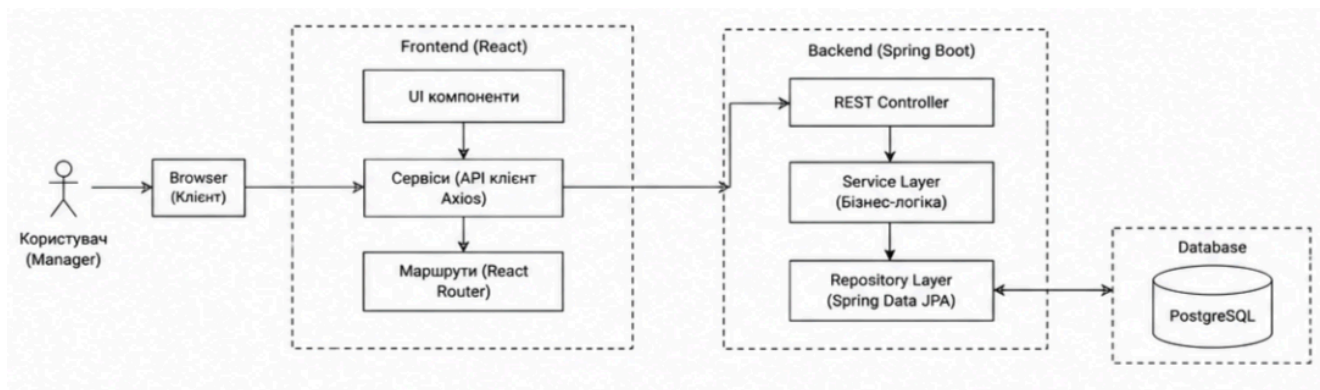


Рис. 3.6 Схема взаємодії frontend, backend та бази даних

Така структура взаємодії дозволяє забезпечити зрозумілий і контрольований обмін даними між компонентами системи. Окреме виділення API також спрощує подальше розширення ServiceControl. Наприклад, у майбутньому до того самого backend API можна підключити мобільний застосунок для майстрів або окремий клієнтський портал для запису на ремонт [1; 2; 4; 6].

Отже, під час проектування системи ServiceControl було визначено її логічну структуру, рольову модель, структуру бази даних, життєвий цикл заявки, логіку роботи аналітичного модуля Service Health Radar та основні принципи взаємодії frontend, backend і бази даних. Це створює основу для подальшої програмної реалізації системи [7; 8; 9].

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ SERVICECONTROL

4.1 Реалізація авторизації, реєстрації та рольової моделі

Програмна реалізація системи ServiceControl починається з модуля авторизації та реєстрації користувачів. Цей модуль є базовим, оскільки саме він визначає, хто може увійти до системи та які функції будуть доступні після входу. У системі передбачено три основні ролі користувачів: ADMIN, MANAGER та MECHANIC. Кожна роль має окремий набір прав доступу та бачить різний функціонал.

Сторінка входу до системи містить форму авторизації, у якій користувач вводить email та пароль. Після натискання кнопки входу frontend надсилає запит до серверної частини. Backend перевіряє наявність користувача, правильність пароля та активність облікового запису. Якщо дані коректні, користувач отримує доступ до системи відповідно до своєї ролі [7; 8; 9].

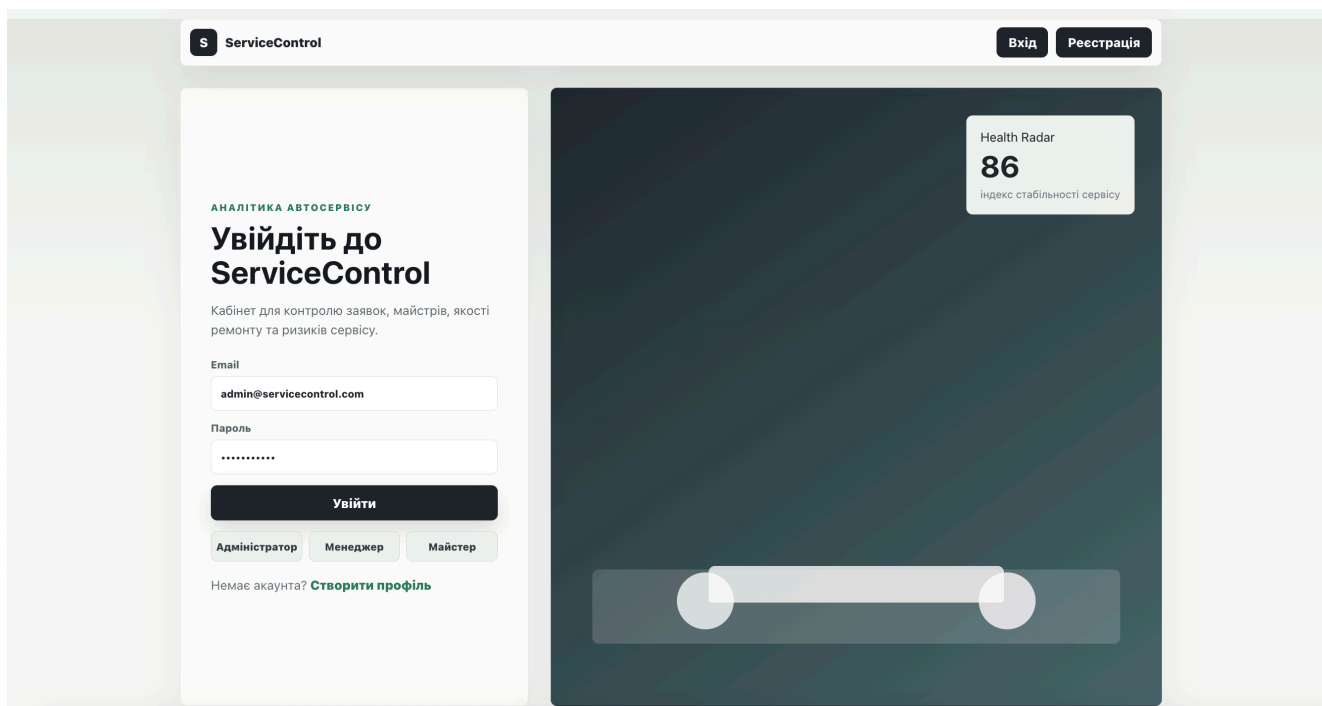


Рис. 4.1 Екранна форма входу до системи ServiceControl

Сторінка реєстрації призначена для створення нового користувача. Вона містить поля для введення імені, email, пароля, телефону, посади та ролі. У навчальній версії системи вибір ролі може бути доступним під час реєстрації, однак у реальному середовищі доцільніше створювати нових користувачів через адміністративний модуль. Це дозволяє уникнути ситуацій, коли звичайний користувач самостійно обирає роль адміністратора.

Рис. 4.2 Екранна форма реєстрації користувача

Після входу до системи користувач перенаправляється на головну сторінку Dashboard. Вміст цієї сторінки залежить від ролі. ADMIN бачить повну аналітику, фінансові показники, завантаженість майстрів, проблемні категорії робіт і Service Health Radar. MANAGER бачить активні заявки, заявки без майстра, клієнтів, автомобілі та базову аналітику. MECHANIC бачить лише власні призначені заявки, строки виконання та статуси робіт [1; 3; 6].

Рольова модель реалізована таким чином, щоб кожен користувач працював тільки з потрібними йому функціями. Це зменшує інформаційне перевантаження, підвищує безпеку системи та знижує ризик випадкового редагування чужих даних.

Після авторизації користувач також може перейти на сторінку профілю. У профілі відображаються його ім'я, email, телефон, посада, роль і додаткова інформація. Для ролі MECHANIC профіль може також показувати спеціалізацію майстра та пов'язані з ним заявки [1; 2; 4; 6].

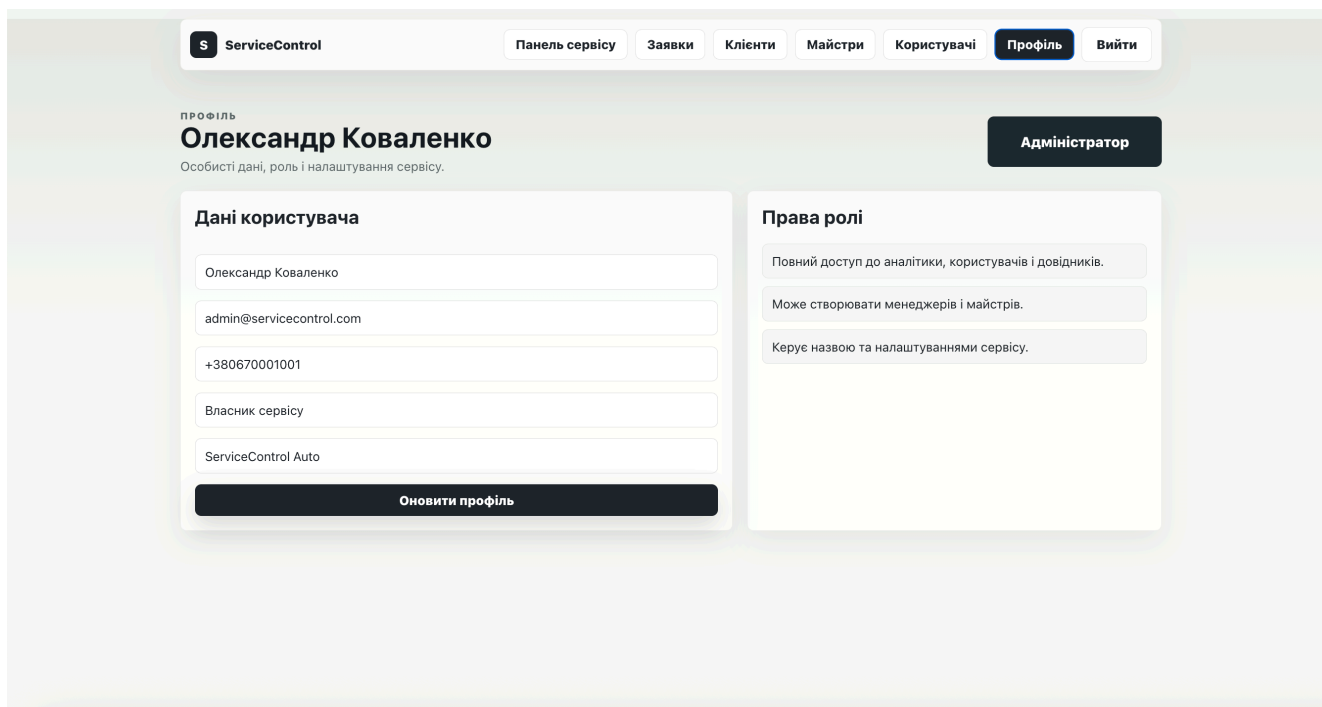


Рис. 4.3 Сторінка профілю користувача системи ServiceControl

Таким чином, модуль авторизації та реєстрації забезпечує контроль доступу до системи, а рольова модель дозволяє адаптувати інтерфейс ServiceControl під реальні задачі адміністратора, менеджера та майстра автосервісу [1; 2; 4; 6].

4.2 Реалізація головної сторінки Dashboard та Service Health Radar

Головна сторінка Dashboard є центральною аналітичною сторінкою системи ServiceControl. Вона призначена для швидкого перегляду поточного стану автосервісу. На цій сторінці користувач може побачити основні показники роботи сервісу, проблемні зони та рекомендації системи [1; 3; 6].

Для користувача з роллю ADMIN Dashboard містить найбільш повний набір показників. До них належать:

- загальний індекс стану автосервісу;
- кількість активних заявок;
- кількість завершених заявок;
- кількість прострочених заявок;
- кількість заявок без призначеного майстра;
- середня оцінка клієнтів;
- загальна виручка;
- завантаженість майстрів;
- проблемні категорії ремонтних робіт;
- рекомендації Service Health Radar.

Для користувача з роллю MANAGER Dashboard має більш прикладний характер. Він показує активні заявки, заявки на поточний день, заявки без майстра, клієнтів, автомобілі та базові показники виконання робіт. Фінансова або адміністративна аналітика для менеджера може бути обмежена [1; 3; 6].

Для користувача з роллю MECHANIC головна сторінка є спрощеною. Вона відображає тільки ті заявки, які призначені конкретному майстру. Також можуть показуватися найближчі дедлайни, статуси поточних робіт і кількість виконаних заявок за певний період [1; 2; 4; 6].

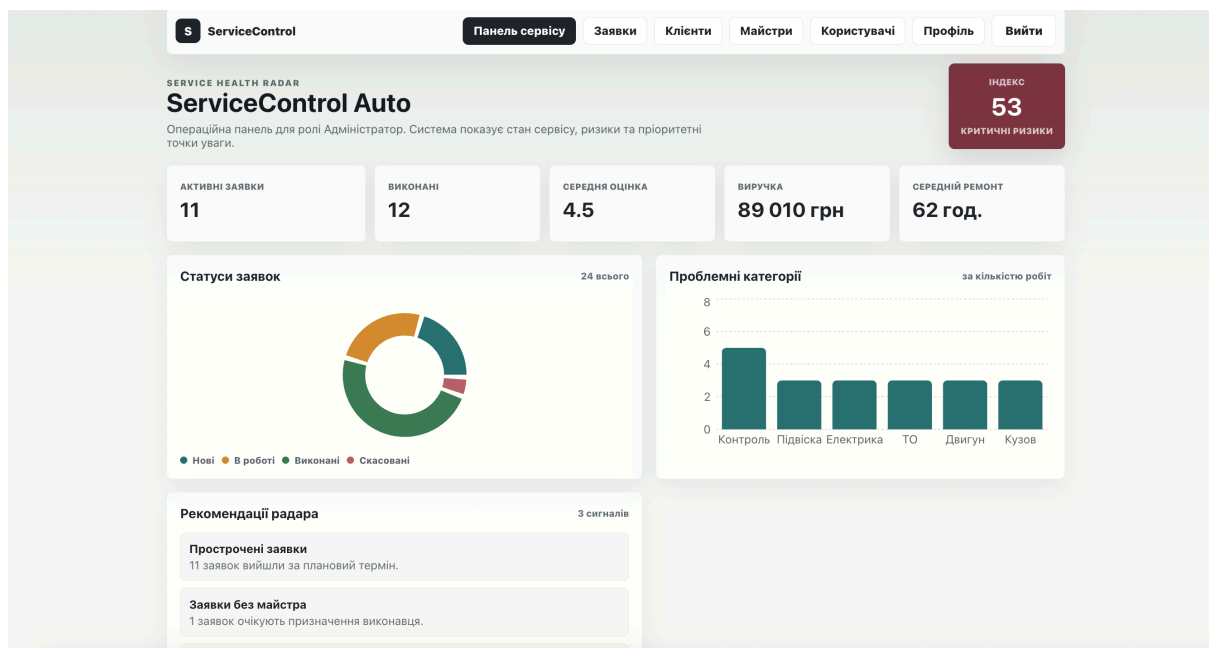


Рис. 4.4 Головна сторінка адміністратора з аналітичними показниками

Особливою частиною Dashboard є блок Service Health Radar. Його завдання полягає в тому, щоб не просто показувати окремі статистичні значення, а формувати загальну оцінку стану автосервісу. Цей показник може відображатися у вигляді значення від 0 до 100 [1; 3; 6]. Наприклад, якщо система показує Service Health 82/100, це означає, що загальний стан сервісу є стабільним, але є окремі зони, які потребують уваги. Якщо показник знижується, адміністратор може швидко побачити причини: прострочені заявки, перевантаження майстрів, низькі оцінки клієнтів або часті затримки в певній категорії ремонту [1; 2; 4; 6].

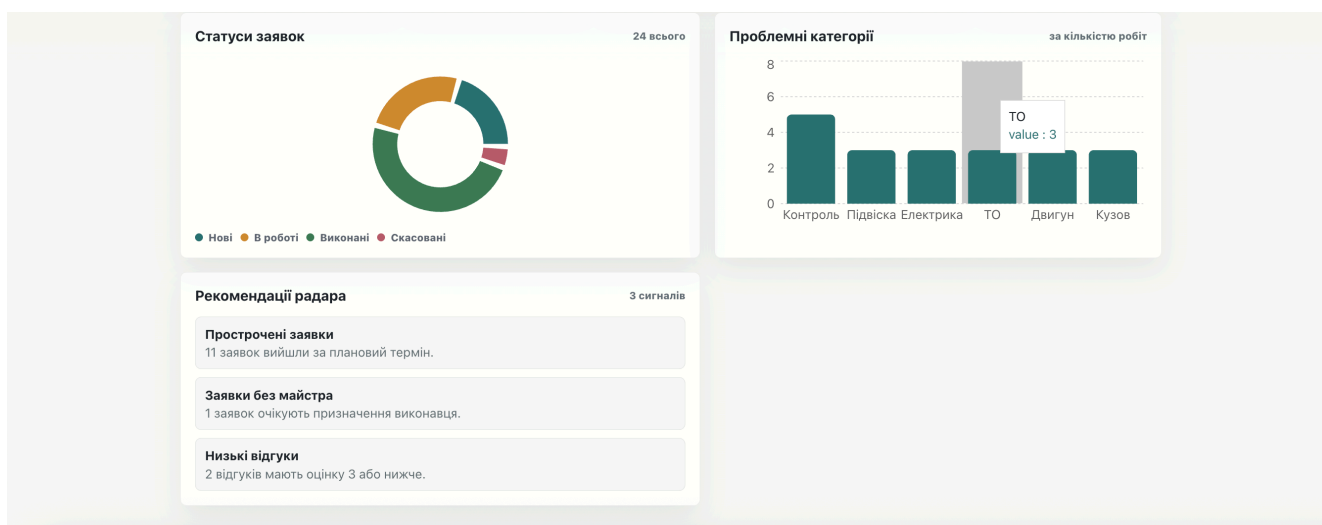


Рис. 4.5 Блок Service Health Radar у системі ServiceControl

Service Health Radar аналізує декілька груп даних:

- заявки на ремонт;
- строки виконання заявок;
- призначення майстрів;
- кількість активних заявок у кожного майстра;
- виконані роботи;
- категорії ремонтів;
- відгуки клієнтів;
- середню оцінку обслуговування.

На основі цих даних система формує рекомендації. Наприклад:

- «2 заявки потребують уваги»;
- «Майстер Ігор Литвин має підвищене навантаження»;
- «Категорія “Підвіска” найчастіше спричиняє затримки»;
- «Середня оцінка клієнтів знизилася до 4.1»;
- «Є заявки без призначеного майстра».

Завдяки цьому Dashboard стає не просто сторінкою зі статистикою, а інструментом підтримки управлінських рішень. Адміністратор може швидко визначити, які процеси потребують втручання, а менеджер — оперативно реагувати на заявки, що залишилися без виконавця або наближаються до дедлайну [1; 3; 6].

4.3 Реалізація модуля заявок та виконаних робіт

Модуль заявок є основним робочим модулем системи ServiceControl. Саме через нього відбувається створення, редагування, контроль і завершення ремонтних робіт. Заявка поєднує між собою клієнта, автомобіль, майстра, опис проблеми, статус, пріоритет, строки виконання, виконані роботи, вартість і відгук клієнта [1; 2; 4; 6].

Сторінка заявок містить список усіх ремонтних заявок, доступних поточному користувачу. Для ADMIN і MANAGER відображається повний або розширений список заявок автосервісу. Для MECHANIC відображаються лише ті заявки, які призначені конкретному майстру [1; 2; 4; 6].

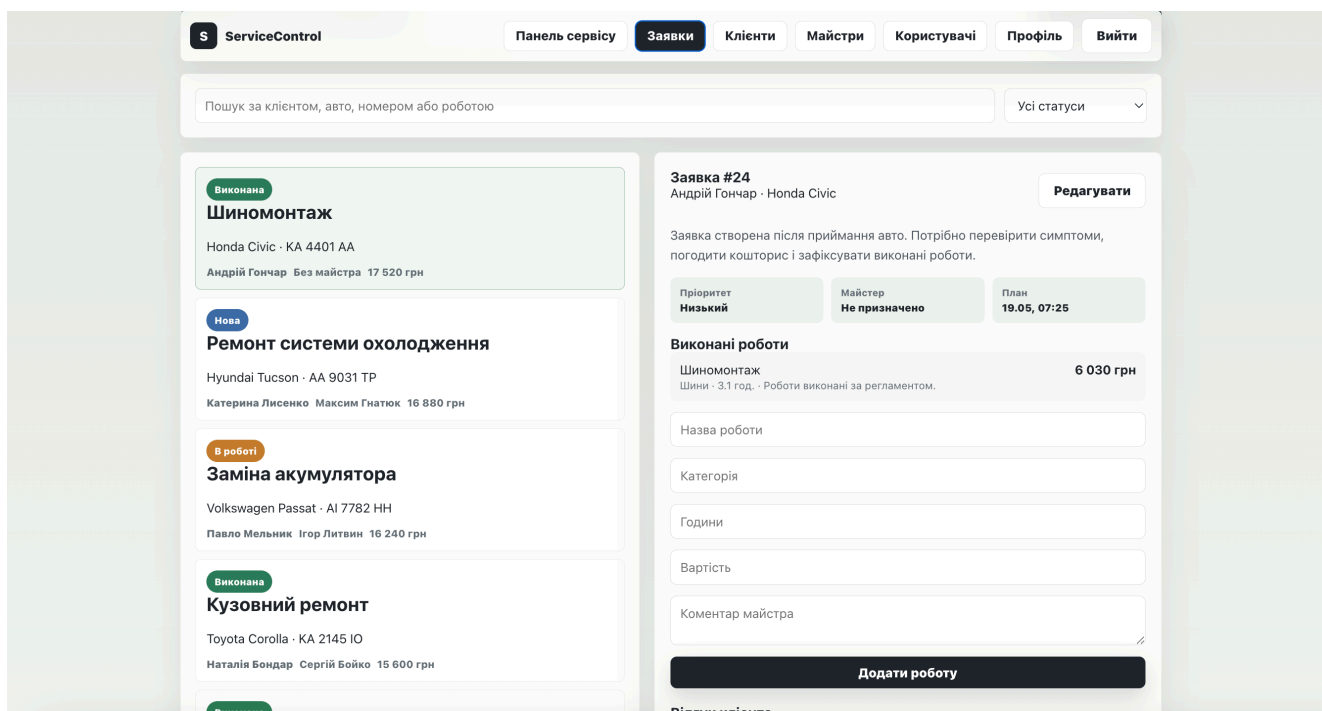


Рис. 4.6 Сторінка керування заявками на ремонт

Під час створення заявки користувач обирає клієнта та автомобіль, вказує тип ремонту, опис проблеми, пріоритет і планову дату завершення. Якщо майстер уже відомий, його можна призначити одразу. Якщо ні, заявка зберігається без виконавця та відображається в аналітиці як така, що потребує уваги [1; 2; 4; 6].

Сторінка деталей заявки містить повну інформацію про ремонт. У ній відображаються клієнт, автомобіль, майстер, опис проблеми, статус, пріоритет, дата відкриття, планова дата завершення, фактична дата завершення, список виконаних робіт, загальна вартість і відгук клієнта [1; 2; 4; 6].

Заявка #21

Наталія Бондар · Toyota Corolla

Редагувати

Заявка створена після приймання авто. Потрібно перевірити симптоми, погодити кошторис і зафіксувати виконані роботи.

Пріоритет
Звичайний

Майстер
Сергій Бойко

План
14.05, 07:25

Виконані роботи

Кузовний ремонт

Гальма · 1 год. · Роботи виконані за регламентом.

5 400 грн

Контроль якості

Контроль · 0.5 год. · Фінальний огляд без критичних зауважень.

450 грн

Назва роботи

Категорія

Години

Вартість

Коментар майстра

Додати роботу

Відгук клієнта

Рис. 4.7 Форма перегляду деталей заявки на ремонт

Для майстра сторінка деталей заявки є основною робочою сторінкою. Він може переглянути опис проблеми, поточний статус, строк виконання та додати виконані роботи. Під час додавання роботи майстер вказує назву, категорію, витрачений час, вартість і коментар [1; 2; 4; 6].

Редагувати заявку

Заявка #21
Наталія Бондар · Toyota Corolla

Клієнт: Наталія Бондар

Автомобіль: Оберіть

Майстер: Сергій Бойко

Статус: Виконана

Пріоритет: Звичайний

Вартість: 15600

План: 14.05.2026, 07:25

Факт: 14.05.2026, 07:25

Тип робіт: Кузовний ремонт

Опис: Заявка створена після приймання авто. Потрібно перевірити симптоми, погодити кошторис і зафіксувати виконані роботи.

Скасувати Зберегти

Додати роботу

Рис. 4.8 Форма додавання виконаної роботи до заявки

Після додавання виконаних робіт система може оновлювати загальну вартість заявки. Це дозволяє бачити не тільки список робіт, а й фінальний підсумок ремонту. Дані про виконані роботи також використовуються в аналітичному модулі для визначення проблемних категорій і середньої тривалості ремонту [1; 3; 6].

Важливою частиною модуля заявок є історія подій. Коли користувач створює заявку, змінює статус, призначає майстра, додає роботу або завершує ремонт,

система може зберігати відповідний запис у таблиці `order_events`. Це дозволяє відстежити послідовність змін і підвищує прозорість роботи автосервісу [1; 2; 4; 6].

Таким чином, модуль заявок забезпечує повний цикл роботи з ремонтом: від створення заявки до завершення, фіксації виконаних робіт, розрахунку вартості та отримання відгуку клієнта [1; 2; 4; 6].

4.4 Реалізація довідників клієнтів, автомобілів і майстрів

Для повноцінної роботи автосервісу система ServiceControl містить довідники клієнтів, автомобілів і майстрів. Ці модулі забезпечують збереження структурованої інформації, яка використовується під час створення заявок і формування аналітики [1; 2; 4; 6].

Модуль клієнтів призначений для ведення бази клієнтів автосервісу. На сторінці клієнтів відображається список осіб, які зверталися до сервісу. Для кожного клієнта зберігаються ім'я, телефон, email, нотатки та дата створення запису [1; 2; 4; 6].

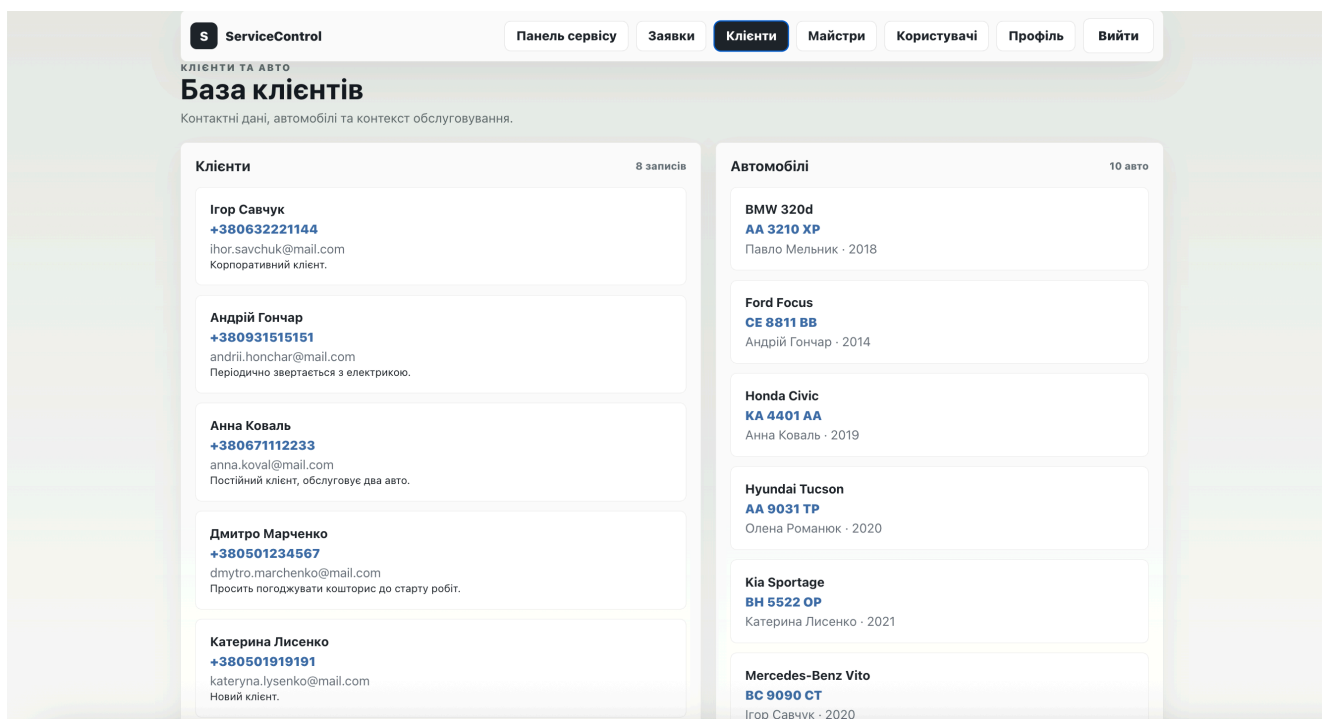


Рис. 4.9 Сторінка керування клієнтами автосервісу

Модуль майстрів використовується для ведення списку працівників, які виконують ремонтні роботи. Для кожного майстра зберігаються ім'я, спеціалізація, телефон, погодинна ставка, активність і, за потреби, зв'язок з обліковим записом користувача. Це дозволяє системі визначати, які заявки повинен бачити майстер після входу [1; 2; 4; 6].

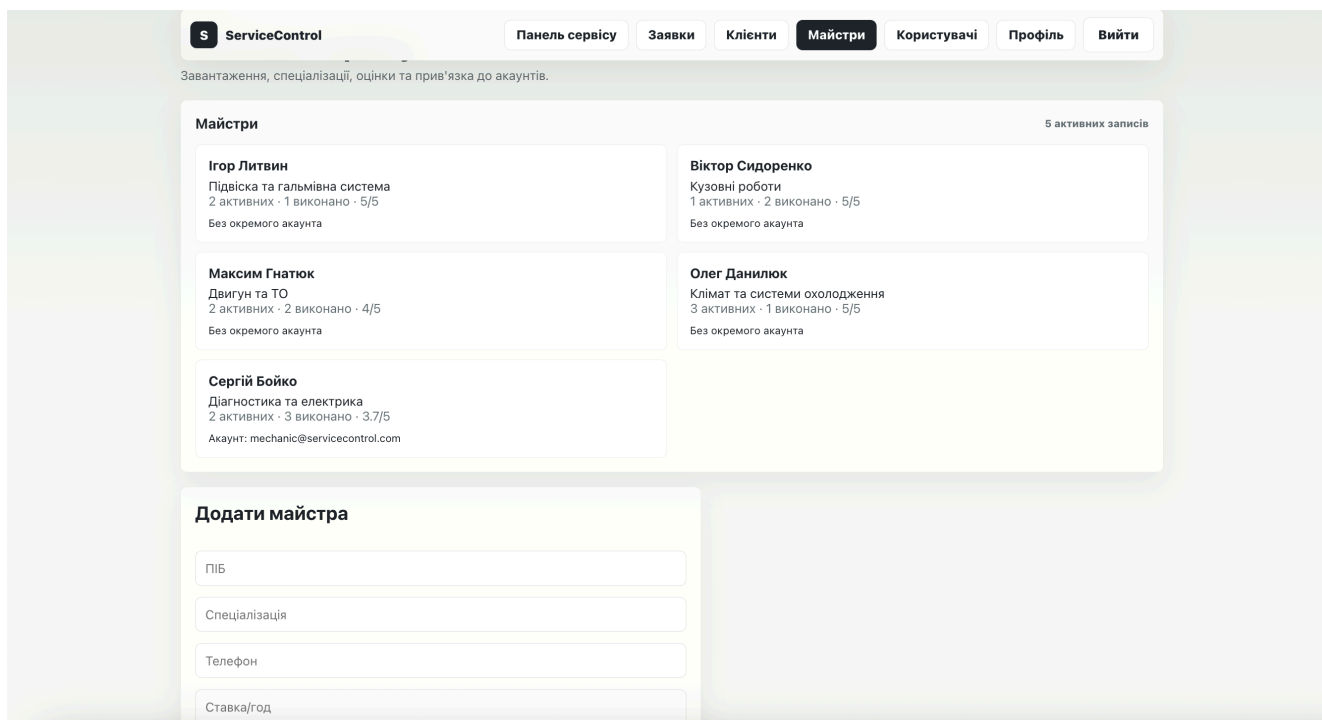


Рис. 4.10 Сторінка керування майстрами автосервісу

Для ADMIN модуль майстрів є повністю доступним. Адміністратор може створювати, редагувати або деактивувати майстрів. MANAGER може переглядати майстрів і призначати їх на заявки. MECHANIC не має доступу до загального списку майстрів, оскільки для нього ця інформація не є необхідною [1; 2; 4; 6].

Довідники клієнтів, автомобілів і майстрів є важливою основою для роботи всієї системи. Вони дозволяють уникнути дублювання даних, швидко створювати заявки, переглядати історію ремонтів і формувати аналітичні показники [1; 3; 6].

4.5 Реалізація адміністративного модуля користувачів

Адміністративний модуль користувачів доступний тільки для ролі ADMIN. Він призначений для керування обліковими записами працівників автосервісу. Через цей модуль адміністратор може контролювати, хто має доступ до системи та які права має кожен користувач [1; 2; 4; 6].

На сторінці користувачів відображається список облікових записів. Для кожного користувача показуються ім'я, email, роль, телефон, посада, активність акаунта та дата створення.

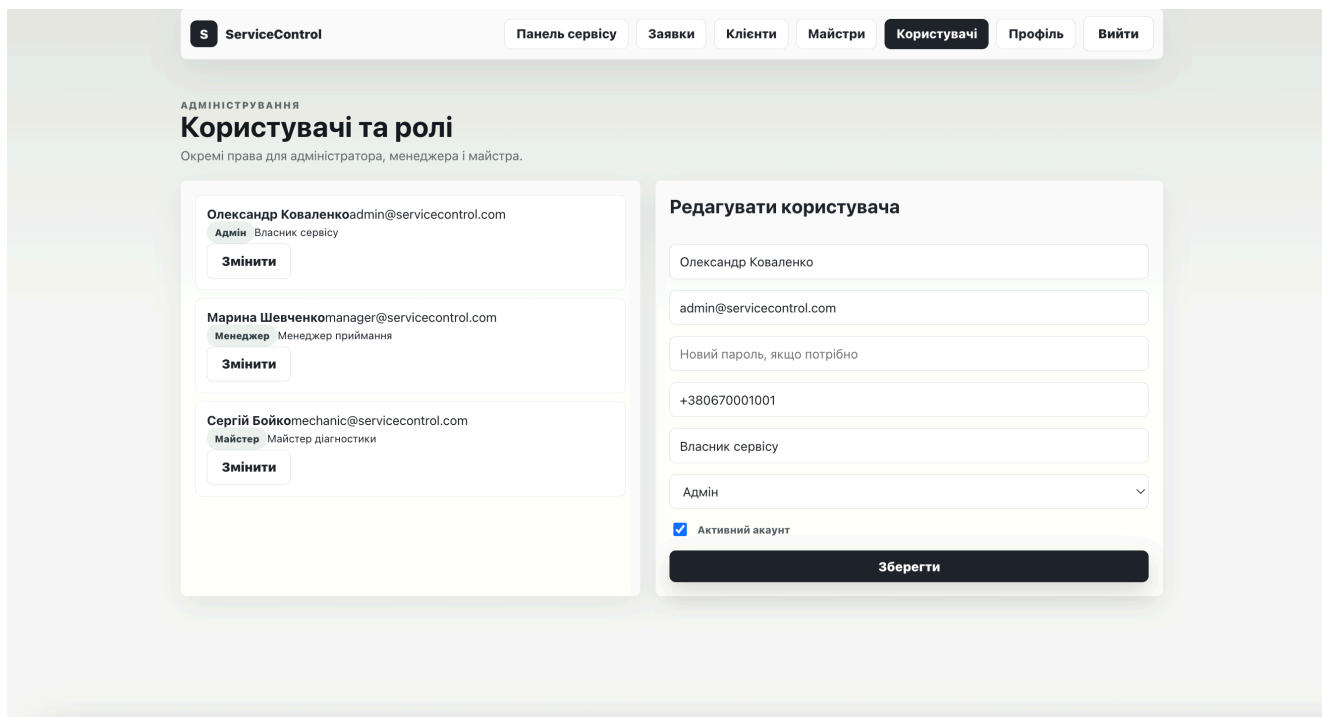


Рис. 4.11 Сторінка керування користувачами системи ServiceControl

Реалізація цього модуля є важливою для безпеки системи. Якщо працівник більше не працює в автосервісі, адміністратор може деактивувати його обліковий запис. Якщо користувач змінює посаду, адміністратор може змінити його роль. Наприклад, працівник може перейти з ролі MANAGER на ADMIN або отримати роль MECHANIC [1; 2; 4; 6].

5 ТЕСТУВАННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ РОЗРОБКИ

5.1 Мета та підхід до тестування системи

Після завершення основної програмної реалізації системи ServiceControl було проведено тестування її функціональних можливостей. Метою тестування є перевірка правильності роботи основних модулів системи, відповідності реалізованого функціоналу поставленим вимогам, коректності рольового доступу, стабільності роботи з даними та зручності використання інтерфейсу [7; 8; 9].

Оскільки система ServiceControl складається з клієнтської частини, серверної частини та бази даних, тестування виконувалося на декількох рівнях. Перевірялася робота інтерфейсу користувача, коректність виконання запитів до серверної частини, правильність збереження даних у базі, а також логіка відображення інформації залежно від ролі користувача [7; 8; 9].

Тестування проводилося за сценарним підходом. Для кожного модуля було визначено типові дії користувача, очікуваний результат і фактичний результат виконання. Такий підхід дозволяє перевірити систему з позиції реального використання: адміністратор входить у систему, створює користувачів, менеджер створює заявку, майстер додає виконані роботи, а аналітичний модуль формує показники стану автосервісу [7; 8; 9].

Під час тестування перевірялися такі аспекти:

- коректність відкриття сторінок;
- правильність авторизації користувачів;
- відповідність доступних сторінок ролі користувача;
- створення, редагування та пошук записів;
- збереження даних у базі даних;
- зміна статусів заявок;
- призначення майстрів;
- додавання виконаних робіт;

- додавання відгуків клієнтів;
- формування історії подій заявки;
- відображення аналітичних показників;
- коректність роботи Service Health Radar;
- обробка некоректного введення даних.

Для перевірки серверної частини могли використовуватися запити до REST API через інструменти тестування API, а для перевірки клієнтської частини — ручне тестування через веб-інтерфейс. Окремо перевірялася взаємодія системи з базою даних PostgreSQL, оскільки від правильності збереження даних залежить робота всіх аналітичних модулів [19].

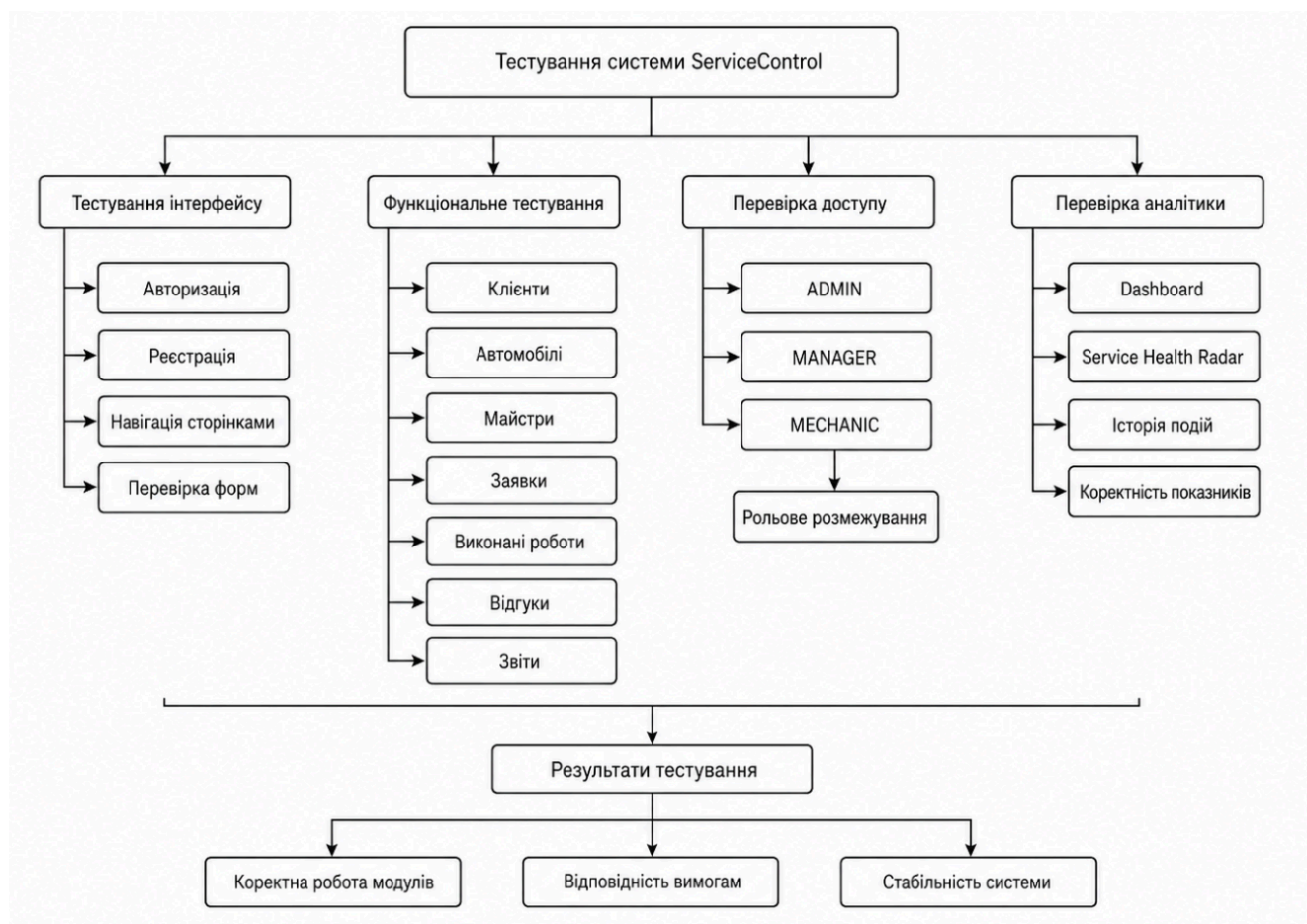


Рис. 5.1 Загальна схема тестування системи ServiceControl

Таким чином, тестування ServiceControl було спрямоване не тільки на пошук технічних помилок, а й на перевірку того, чи відповідає система реальним процесам роботи автосервісу [7; 8; 9].

5.2 Тестування основних функціональних модулів

Функціональне тестування передбачає перевірку того, чи виконує система основні дії, передбачені вимогами. Для ServiceControl такими діями є авторизація користувачів, робота із заявками, клієнтами, автомобілями, майстрами, виконаними роботами, відгуками та аналітичними показниками [7; 8; 9].

Першим етапом було перевірено модуль авторизації. Користувач вводить email і пароль, після чого система перевіряє дані та відкриває доступ до відповідних сторінок. Було перевірено вхід для трьох ролей: ADMIN, MANAGER та MECHANIC. У результаті встановлено, що після входу кожен користувач отримує доступ до свого набору функцій [7; 8; 9].

Далі було протестовано модуль реєстрації. Під час створення нового користувача перевірялося заповнення обов'язкових полів, збереження email, імені, телефону, посади та ролі. Також перевірялася реакція системи на порожні або некоректно заповнені поля [7; 8; 9].

Модуль клієнтів тестувався шляхом створення нового клієнта, редагування його даних, пошуку за іменем, телефоном та email. Після збереження перевірялося, чи відображається новий клієнт у списку та чи може він бути використаний під час створення заявки [7; 8; 9].

Модуль автомобілів перевірявся шляхом додавання автомобіля до конкретного клієнта. Перевірялися поля марки, моделі, номерного знака, VIN-коду, року випуску та власника. Також було перевірено, чи правильно автомобіль відображається в історії клієнта та чи може бути обраний під час створення заявки [7; 8; 9].

Модуль майстрів тестувався через додавання нового майстра, зазначення його спеціалізації, телефону, погодинної ставки та активності. Окремо

перевірялася можливість призначення майстра на заявку, а також відображення його поточного навантаження [7; 8; 9].

Найбільшу увагу було приділено тестуванню модуля заявок. Перевірялися створення нової заявки, вибір клієнта та автомобіля, призначення майстра, встановлення статусу, вибір пріоритету, додавання опису проблеми та планової дати завершення. Після створення заявки перевірялося її відображення у списку, можливість редагування та зміни статусу [7; 8; 9].

Після переведення заявки в роботу було протестовано додавання виконаних робіт. Майстер додавав назву роботи, категорію, кількість витрачених годин, вартість і коментар. Після збереження перевірялося, чи відображається робота в деталях заявки та чи впливає вона на загальну вартість ремонту [7; 8; 9].

Модуль відгуків тестувався після завершення заявки. Перевірялося додавання оцінки клієнта та текстового коментаря. Дані відгуку повинні зберігатися в системі та враховуватися під час формування середньої оцінки клієнтів на Dashboard [1; 3; 6].

Окремо було протестовано історію подій заявки. Після створення заявки, призначення майстра, зміни статусу, додавання роботи та відгуку система повинна фіксувати відповідні події. Це дозволяє відстежити повний шлях заявки від створення до завершення [1; 2; 4; 6].

Основні тестові сценарії наведено в таблиці 5.1.

Таблиця 5.1

Тестові сценарії перевірки системи ServiceControl

№	Модуль	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
1	Авторизація	Вхід користувача ADMIN з коректними даними	Відкрито Dashboard адміністратора	Відкрито Dashboard адміністратора	Успішно
2	Авторизація	Вхід користувача з некоректним паролем	Система показує повідомлення про помилку	Повідомлення відображено	Успішно

Тестові сценарії перевірки системи ServiceControl

№	Модуль	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
3	Реєстрація	Створення нового користувача	Користувача додано до системи	Користувача додано	Успішно
4	Клієнти	Додавання нового клієнта	Клієнт збережений і відображається у списку	Клієнт відображається	Успішно
5	Автомобілі	Додавання автомобіля до клієнта	Автомобіль прив'язано до клієнта	Автомобіль збережено	Успішно
6	Майстри	Додавання нового майстра	Майстер збережений у довіднику	Майстер відображається у списку	Успішно
7	Заявки	Створення нової заявки	Заявка створена зі статусом NEW	Заявка створена	Успішно
8	Заявки	Призначення майстра на заявку	Майстра призначено	Майстер відображається в заявці	Успішно
9	Заявки	Зміна статусу на IN PROGRESS	Статус заявки оновлено	Статус змінено	Успішно
10	Виконані роботи	Додавання виконаної роботи до заявки	Робота додана до заявки	Робота відображається в деталях	Успішно
11	Заявки	Завершення заявки	Статус змінено на DONE	Заявку завершено	Успішно
12	Відгуки	Додавання відгуку клієнта	Оцінка та коментар збережені	Відгук відображається	Успішно
13	Події заявки	Перегляд історії змін	Система показує події заявки	Події відображаються	Успішно
14	Dashboard	Перегляд основних показників	Показники відображаються на головній сторінці	Показники відображено	Успішно
15	Service Health Radar	Перевірка формування індексу	Система формує індекс стану сервісу	Індекс сформовано	Успішно

Результати тестування показали, що основні функціональні модулі системи працюють коректно. Реалізовані функції відповідають поставленим вимогам і дозволяють виконувати повний цикл роботи автосервісу: від створення клієнта та заявки до завершення ремонту, додавання відгуку й аналізу показників [7; 8; 9].

5.3 Перевірка рольового доступу

Окремим етапом тестування була перевірка рольового доступу. Для системи ServiceControl це особливо важливо, оскільки різні категорії користувачів мають різні права та повинні бачити різний набір сторінок [7; 8; 9].

Користувач з роллю ADMIN повинен мати повний доступ до системи. Під час тестування було перевірено, що адміністратор може відкривати Dashboard, керувати користувачами, клієнтами, автомобілями, майстрами, заявками, виконаними роботами, відгуками, звітами та аналітичним модулем Service Health Radar [7; 8; 9].

Користувач з роллю MANAGER повинен мати доступ до робочих процесів автосервісу, але не до повного адміністративного управління. Під час тестування було перевірено, що менеджер може створювати та редагувати заявки, додавати клієнтів і автомобілі, переглядати майстрів, призначати майстрів на заявки, додавати відгуки та переглядати базову аналітику. Водночас сторінка управління користувачами для нього недоступна [7; 8; 9].

Користувач з роллю MECHANIC повинен мати найбільш обмежений доступ. Під час тестування було перевірено, що майстер бачить лише власні призначені заявки, може переглядати їх деталі та додавати виконані роботи. Він не має доступу до сторінок користувачів, клієнтів, повної аналітики, фінансових показників і загального списку всіх заявок [7; 8; 9].

Перевірка рольового доступу наведена в таблиці 5.2.

Перевірка прав доступу користувачів

Функція системи	ADMIN	MANAGER	MECHANIC
Вхід до системи	+	+	+
Перегляд профілю	+	+	+
Перегляд повного Dashboard	+	-	-
Перегляд базового Dashboard	+	+	-
Перегляд власних заявок	+	+	+
Перегляд усіх заявок	+	+	-
Створення заявки	+	+	-
Редагування заявки	+	+	Частково
Призначення майстра	+	+	-
Додавання виконаних робіт	+	+	+
Додавання відгуку клієнта	+	+	-
Управління клієнтами	+	+	-
Управління автомобілями	+	+	-
Управління майстрами	+	Частково	-
Управління користувачами	+	-	-
Перегляд Service Health Radar	+	Частково	-
Перегляд звітів	+	Частково	-

У таблиці знак «+» означає повний доступ до функції, знак «-» означає відсутність доступу, а позначка «Частково» означає, що користувач має обмежені права відповідно до своєї ролі.

5.4 Перевірка роботи аналітичного модуля Service Health Radar

Аналітичний модуль Service Health Radar є однією з ключових особливостей системи ServiceControl, тому його тестування було виділено в окремий етап. Метою

перевірки було встановити, чи правильно система аналізує поточні дані автосервісу та формує загальний індекс стану сервісу [7; 8; 9].

У першому сценарії було створено заявку з плановою датою завершення, яка вже минула, але статус заявки залишався NEW або IN_PROGRESS. Очікуваний результат — система повинна визначити таку заявку як прострочену та знизити загальний індекс Service Health. Під час тестування прострочена заявка була визначена коректно [7; 8; 9].

У другому сценарії було створено заявку без призначеного майстра. Очікувалося, що система покаже її як заявку, що потребує уваги. У результаті Service Health Radar сформував відповідне попередження [1; 3; 6].

У третьому сценарії одному майстру було призначено більшу кількість активних заявок порівняно з іншими. Система повинна була визначити підвищене навантаження та вивести рекомендацію щодо перерозподілу робіт. Під час тестування перевантаження майстра було відображено в аналітичному блоці [7; 8; 9].

У четвертому сценарії було додано декілька відгуків із низькою оцінкою. Очікуваний результат — зниження середньої оцінки клієнтів і вплив на загальний індекс стану сервісу. Система коректно врахувала ці дані під час формування Service Health [1; 2; 4; 6].

У п'ятому сценарії було перевірено визначення проблемних категорій робіт. Для цього в системі були створені заявки з роботами однієї категорії, які часто завершувалися із затримкою. Система повинна була позначити цю категорію як проблемну. У результаті відповідна категорія була відображена серед ризикових [7; 8; 9].

Приклад результату роботи Service Health Radar може мати такий вигляд:

- Service Health: 82/100;
- 2 заявки потребують уваги;
- один майстер має підвищене навантаження;
- категорія «Підвіска» найчастіше спричиняє затримки;
- середня оцінка клієнтів знизилася до 4.1.

Результати тестування показали, що Service Health Radar виконує свою основну задачу: не тільки відображає окремі показники, а й допомагає швидко виявити проблемні процеси в роботі автосервісу. Завдяки цьому система може використовуватися як інструмент підтримки управлінських рішень [7; 8; 9].

5.5 Оцінка відповідності системи поставленим вимогам

Після функціонального тестування та перевірки рольового доступу було виконано оцінку відповідності реалізованої системи поставленим вимогам. На цьому етапі перевірялося, чи були реалізовані основні функціональні та нефункціональні вимоги, визначені в першому розділі роботи [7; 8; 9].

Оцінка відповідності показала, що система ServiceControl реалізує основні можливості, необхідні для управління роботою автосервісу. Вона забезпечує авторизацію користувачів, рольове розмежування доступу, роботу з клієнтами, автомобілями, майстрами, заявками, виконаними роботами, відгуками, історією подій, Dashboard, Service Health Radar і звітами [1; 3; 6].

Відповідність системи поставленим вимогам наведено в таблиці 5.3.

Таблиця 5.3

Відповідність реалізованої системи поставленим вимогам

Вимога	Реалізація в системі	Статус
Авторизація користувачів	Реалізовано вхід за email та паролем	Виконано
Реєстрація користувачів	Реалізовано форму створення нового користувача	Виконано
Рольовий доступ	Підтримано ролі ADMIN, MANAGER, MECHANIC	Виконано
Управління користувачами	Доступно для ADMIN	Виконано
Управління клієнтами	Реалізовано додавання, редагування, пошук і перегляд клієнтів	Виконано
Управління автомобілями	Реалізовано довідник автомобілів із прив'язкою до клієнтів	Виконано

Відповідність реалізованої системи поставленим вимогам

Вимога	Реалізація в системі	Статус
Управління майстрами	Реалізовано список майстрів, спеціалізацію, ставку та активність	Виконано
Створення заявок	Реалізовано створення заявки на ремонт	Виконано
Редагування заявок	Реалізовано зміну статусу, пріоритету, майстра та опису	Виконано
Додавання виконаних робіт	Реалізовано додавання робіт із часом, вартістю та коментарем	Виконано
Додавання відгуків	Реалізовано оцінку та коментар клієнта	Виконано
Історія подій заявки	Реалізовано фіксацію ключових подій	Виконано
Dashboard	Реалізовано відображення основних показників	Виконано
Service Health Radar	Реалізовано інтегральний індекс стану сервісу	Виконано
Звіти	Реалізовано перегляд звітної інформації	Виконано
Продуктивність	Основні операції виконуються без помітної затримки	Виконано
Зручність інтерфейсу	Інтерфейс поділений за ролями та модулями	Виконано
Збереження даних	Дані зберігаються в PostgreSQL	Виконано
Масштабованість	Архітектура дозволяє додавати нові модулі	Виконано

За результатами оцінки можна зробити висновок, що система відповідає поставленим вимогам і виконує основні задачі, визначені під час аналізу предметної галузі. Реалізоване програмне забезпечення дозволяє централізувати дані автосервісу, спростити роботу персоналу, забезпечити контроль заявок і сформувати аналітичну основу для управління якістю обслуговування [1; 3; 6].

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено веб-систему ServiceControl, призначену для управління та аналізу роботи автосервісу. Система дозволяє централізовано вести облік клієнтів, автомобілів, майстрів, заявок на ремонт, виконаних робіт, відгуків клієнтів та історії змін заявки [1; 2; 4; 6].

У процесі роботи було проведено аналіз предметної галузі, визначено основні проблеми автоматизації автосервісу, зокрема розрізнене зберігання даних, складність контролю строків виконання заявок, нерівномірне навантаження майстрів та відсутність комплексної аналітики якості обслуговування [1; 2; 4; 6].

Було проаналізовано існуючі програмні засоби для автоматизації роботи автосервісів, зокрема AutoLeap, Shopmonkey, AutoFluent та ARI Auto Repair Software. На основі цього аналізу було визначено, що більшість готових рішень орієнтовані переважно на іноземний ринок, мають складну структуру або не забезпечують достатньо зручного аналізу стану сервісу для малого та середнього автосервісу [13].

У роботі було сформовано функціональні та нефункціональні вимоги до системи ServiceControl. Визначено основні ролі користувачів: ADMIN, MANAGER та MECHANIC. Для кожної ролі передбачено окремий набір функцій, що дозволяє розмежувати доступ до даних і зробити інтерфейс зручним для різних категорій працівників автосервісу [1; 2; 4; 6].

Було обрано програмні засоби реалізації системи. Серверну частину реалізовано з використанням Java та Spring Boot, клієнтську частину — з використанням React, TypeScript, Vite та Tailwind CSS. Для збереження даних використано реляційну базу даних PostgreSQL. Такий стек технологій дозволив створити структуровану, масштабовану та зручну для подальшого розвитку веб-систему [22].

У процесі проєктування було розроблено логічну структуру системи, рольову модель доступу, структуру бази даних, процес роботи із заявкою на ремонт, схему

взаємодії frontend, backend та бази даних, а також логіку роботи аналітичного модуля Service Health Radar. Центральною сутністю системи визначено заявку на ремонт, яка об'єднує дані про клієнта, автомобіль, майстра, статус, пріоритет, виконані роботи, вартість, строки виконання та відгук клієнта [7; 8; 9].

Було реалізовано основні модулі системи: авторизацію та реєстрацію користувачів, профіль користувача, Dashboard, управління клієнтами, автомобілями, майстрами, користувачами, заявками, виконаними роботами, відгуками, історією подій і звітами. Особливу увагу приділено аналітичному модулю Service Health Radar, який оцінює загальний стан автосервісу на основі прострочених заявок, завантаженості майстрів, заявок без виконавця, середньої оцінки клієнтів і проблемних категорій робіт [1; 3; 6].

Результати роботи можуть бути використані як основа для впровадження інформаційної системи в малих і середніх автосервісах, які потребують зручного інструменту для централізованого обліку заявок, контролю роботи персоналу, аналізу якості обслуговування та підтримки управлінських рішень [1; 2; 4; 6].

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Сич Б.О., Худік Б.О. Розробка платформи аналізу автосервісів для підвищення якості обслуговування. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року. Збірник тез. – К.: ДУІКТ, 2025, С.173-175.
2. Сич Б.О., Задонцев Ю.В. Розробка системи аналізу автосервісу для підвищення якості обслуговування. Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року. Збірник тез. К.: ДУІКТ, 2026 (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. CRM для автосервісу, автомагазину чи СТО: що обрати в Україні. KeyCRM Blog. URL: <https://blog.keycrm.app/uk/crm-dlya-avtoservisu-avtomagazinu-chi-sto-shho-obrati-v-ukraini/>.
2. Автоматизація автосервісу: CRM для СТО. ToDo LTD. URL: <https://todo.ltd/moduli/crm/programa-dlya-sto-ta-avtoservisiv/>.
3. CRM система для автомобільного бізнесу. KeepinCRM. URL: <https://keepincrm.com/crm-for-car-showroom-and-service>.
4. CRM для автосервісу та СТО: як працює автоматизація автосервісу. WEZOM. URL: <https://wezom.com.ua/ua/blog/crm-dlja-avtoservisov-i-sto>.
5. Найкращі CRM-системи для СТО. Shelfy. URL: <https://shelfy.com.ua/categories/crm-systems/service-station/>.
6. CRM система для автосервісу в Києві і Україні. ITUA. URL: <https://itua.com.ua/crm-dlja-avtoservisu-ua>.
7. Збірник матеріалів II Міжнародної науково-практичної конференції. Державний університет інформаційно-комунікаційних технологій. 2021. URL: https://duikt.edu.ua/uploads/n_9074_59003267.pdf.
8. Проблеми комп'ютерної інженерії: збірник тез. Державний університет інформаційно-комунікаційних технологій. 2021. URL: https://duikt.edu.ua/uploads/n_9941_63776971.pdf.
9. Збірник матеріалів науково-практичної конференції. Державний університет інформаційно-комунікаційних технологій. 2022. URL: https://duikt.edu.ua/uploads/p_2121_99261151.pdf.
10. AutoLeap: офіційний сайт програмного забезпечення для керування автосервісом. URL: <https://autoleap.com/>.
11. Shopmonkey: офіційний сайт програмного забезпечення для керування автосервісом. URL: <https://www.shopmonkey.io/>.

- 12.AutoFluent: офіційний сайт програмного забезпечення для керування автосервісом. URL: <https://www.autofluent.com/>.
- 13.ARI Auto Repair Software: офіційний сайт програмного забезпечення для автосервісів. URL: <https://ari.app/>.
- 14.Java Documentation. Object-Oriented Programming Concepts. Oracle. URL: <https://docs.oracle.com/javase/tutorial/java/concepts/index.html>.
- 15.Spring Boot Reference Documentation. Spring. URL: <https://docs.spring.io/spring-boot/>.
- 16.Spring Security Reference Documentation. Spring. URL: <https://docs.spring.io/spring-security/reference/>.
- 17.Spring Data JPA Reference Documentation. Spring. URL: <https://docs.spring.io/spring-data/jpa/reference/>.
- 18.Hibernate ORM Documentation. Hibernate. URL: <https://hibernate.org/orm/documentation/>.
- 19.PostgreSQL Documentation. PostgreSQL Global Development Group. URL: <https://www.postgresql.org/docs/>.
- 20.React Documentation. Meta Open Source. URL: <https://react.dev/>.
- 21.TypeScript Documentation. Microsoft. URL: <https://www.typescriptlang.org/docs/>.
- 22.Vite Documentation. URL: <https://vite.dev/guide/>.
- 23.Docker Documentation. Docker Inc. URL: <https://docs.docker.com/>.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Розробка системи аналізу автосервісу для підвищення якості обслуговування

Виконав студент 4 курсу
групи ПД-44
Сич Богдан Олегович
Керівник роботи
к.т.н, доц. Задонцев Юрій Вікторович

Київ - 2026

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну галузь управління автосервісом та визначити проблеми, що впливають на якість обслуговування клієнтів.
2. Дослідити існуючі програмні засоби для автоматизації автосервісів, їх можливості щодо контролю заявок, роботи майстрів та оцінювання сервісу.
3. Сформулювати функціональні та нефункціональні вимоги до web-системи ServiceControl.
4. Дослідити та на основі аналізу обґрунтувати вибір технологій для реалізації клієнтської частини, серверної частини, бази даних та розгортання системи.
5. Спроектувати логічну структуру ServiceControl, рольову модель доступу, базу даних та основні алгоритми аналізу показників роботи автосервісу.
6. Реалізувати основні модулі системи: авторизацію, профіль користувача, управління клієнтами, автомобілями, майстрами та заявками.
7. Розробити механізм фіксації виконаних робіт, відгуків клієнтів, історії подій заявки та показників якості обслуговування.
8. Реалізувати аналітичний модуль Service Health Radar для оцінювання стану автосервісу.
9. Провести тестування web-системи ServiceControl та проаналізувати результати перевірки основних функцій.

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення аналізу роботи автосервісу за рахунок розробки web-системи ServiceControl для централізованого обліку заявок, клієнтів, автомобілів, майстрів та виконаних робіт.

Об'єкт дослідження: процеси управління заявками, ремонтними роботами та якістю роботи автосервісу.

Предмет дослідження: засоби, методи та технології реалізації web-системи ServiceControl для автоматизації аналізу роботи автосервісу та показників якості обслуговування.

АНАЛІЗ АНАЛОГІВ

<u>Показник</u>	<u>AutoLeap</u>	<u>Shopmonkey</u>	<u>AutoFluent</u>	<u>ARI Auto Repair Software</u>	<u>ServiceControl</u>
Облік клієнтів	+	+	+	+	+
Облік автомобілів	+	+	+	+	+
Створення заявок	+	+	+	+	+
Призначення майстрів	+	+	+	Без аналізу завантаження	+
Контроль строків виконання	+	+	+	Без оцінки ризиків	+
Облік виконаних робіт	+	+	+	+	+
Робота з відгуками клієнтів	+	Обмежені оцінки	Базові відгуки	Базові відгуки	+
Рольовий доступ	+	+	+	Обмежені ролі	+
Аналітична панель	+	+	+	Мінімальні звіти	+
Оцінка завантаженості майстрів	+	+	+	Без детального аналізу	+
Виявлення проблемних категорій робіт	Через звіти	Через звіти	Через звіти	-	+
Інтегральний індекс стану сервісу	-	-	-	-	+
Історія змін заявки	Журнал дій	Журнал дій	Журнал дій	-	+
Орієнтація на український малий/середній автосервіс	-	-	-	Без локальної адаптації	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Авторизація та реєстрація користувачів системи.
2. Розмежування доступу для ролей ADMIN, MANAGER та MECHANIC.
3. Ведення обліку клієнтів автосервісу.
4. Ведення обліку транспортних засобів клієнтів.
5. Створення та перегляд заявок на ремонт.
6. Призначення заявок відповідальним майстрам.
7. Контроль статусу та пріоритету заявки.
8. Фіксація виконаних ремонтних робіт.
9. Додавання оцінок і відгуків клієнтів для аналізу якості обслуговування.
10. Формування аналітичних показників роботи автосервісу та виявлення проблемних зон.

Нефункціональні вимоги

1. Час обробки основних запитів системи — не більше 1–2 секунд.
2. Коректна робота web-застосунку в сучасних браузерях: Google Chrome, Mozilla Firefox, Microsoft Edge.
3. Централізоване зберігання даних у реляційній базі даних PostgreSQL.
4. Захищене збереження паролів користувачів.
5. Підтримка одночасної роботи щонайменше 20 користувачів.
6. Збереження інформації про клієнтів, автомобілі, заявки, майстрів і виконані роботи без втрати даних при стандартних операціях користувача.
7. Можливість подальшого розширення системи новими модулями.

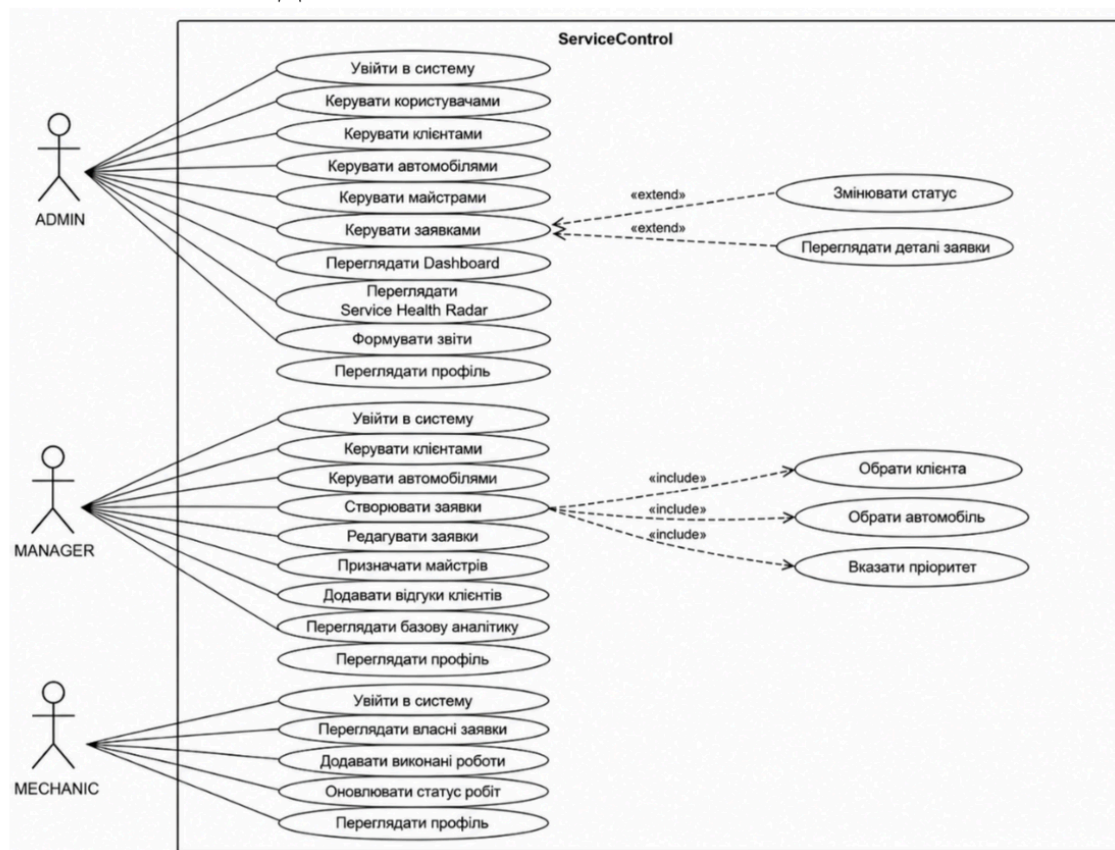
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



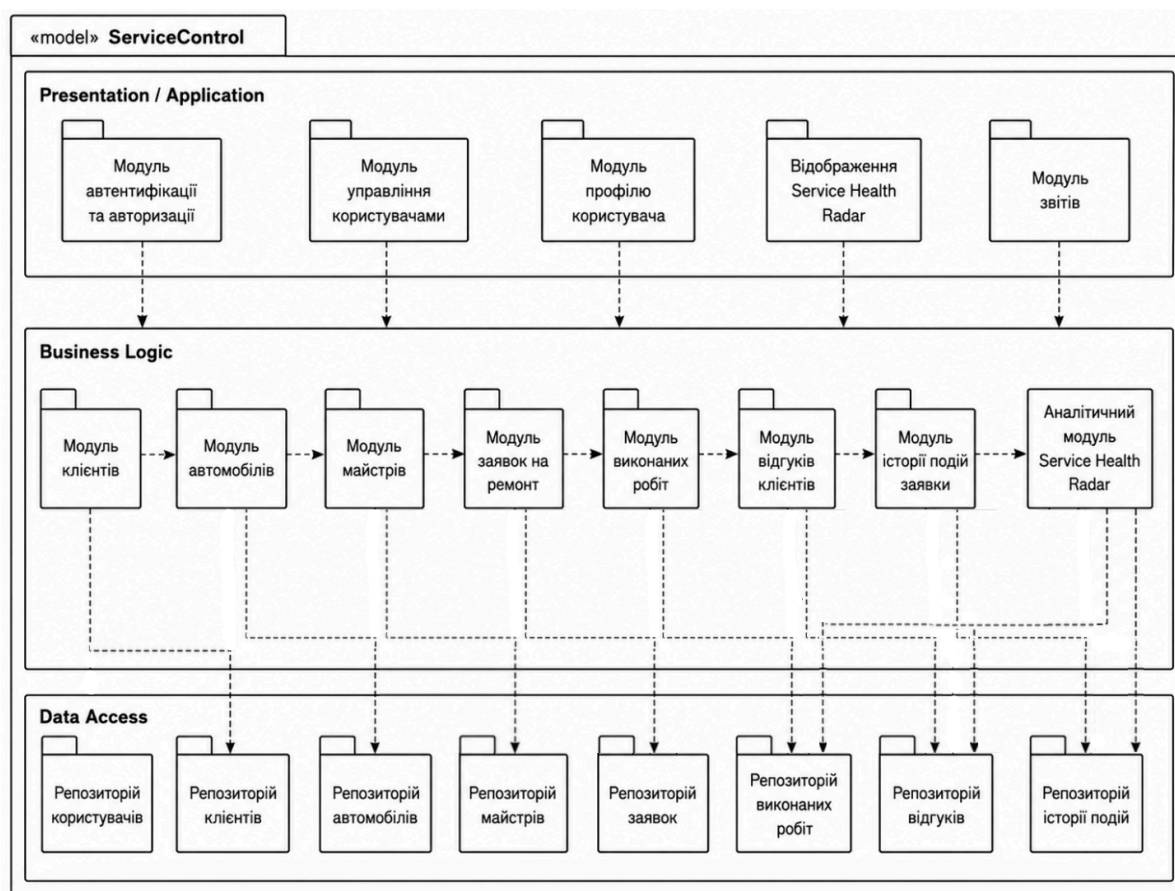
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

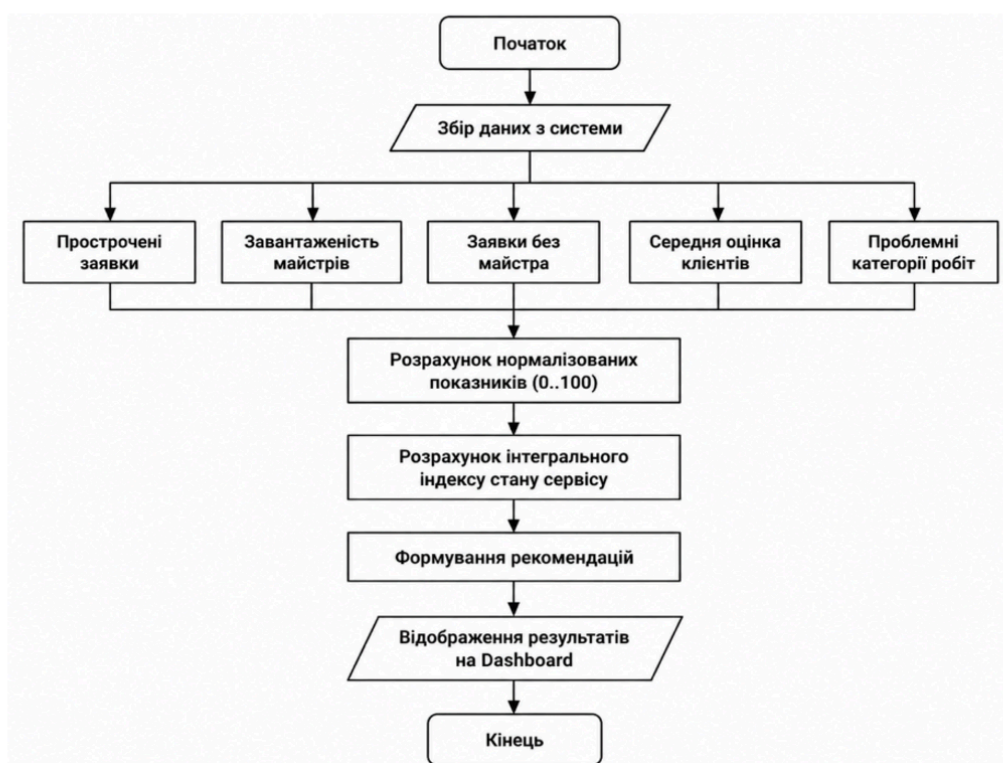


7

ЛОГІЧНА СТРУКТУРА СИСТЕМИ



БЛОК-СХЕМА ЛОГІКИ ФОРМУВАННЯ ІНДЕКСУ SERVICE HEALTH RADAR



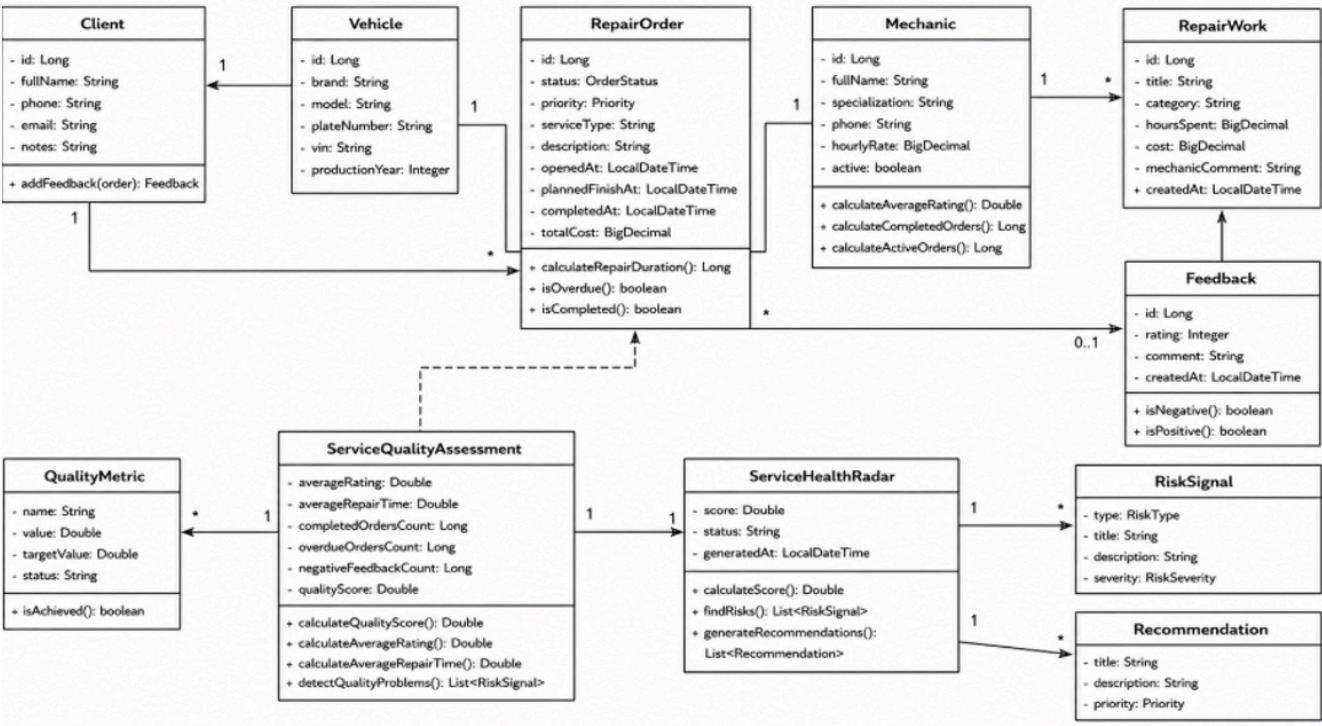
9

РІВНІ ФОРМУВАННЯ ПОКАЗНИКІВ ЯКОСТІ ОБСЛУГОВУВАННЯ

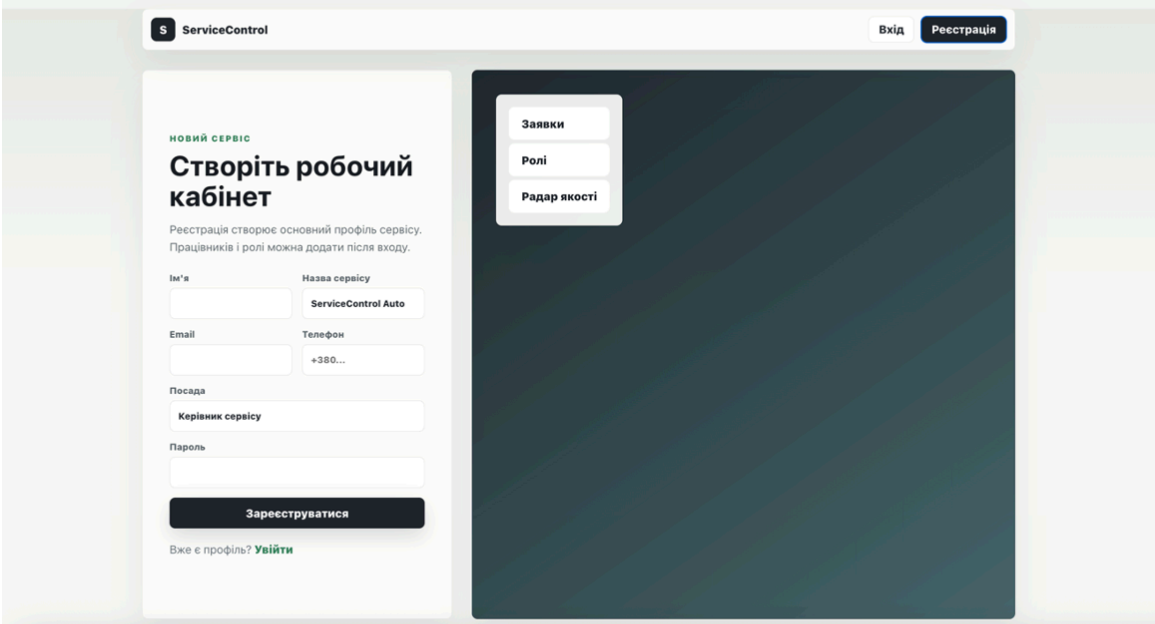
Рівень	Що аналізується	Джерело даних	Значення для якості обслуговування
1. Операційні дані	<u>Заявки, майстри, виконані роботи, відгуки, історія подій</u>	<u>repair orders, mechanics, repair works, feedback, order events</u>	Базові дані, з яких система отримує інформацію про фактичну роботу автосервісу
2. Показники якості	<u>Прострочені заявки, заявки без майстра, завантаженість майстрів, середня оцінка клієнтів, проблемні категорії робіт</u>	Дані з операційних таблиць	Показують, де саме можуть виникати проблеми в обслуговуванні клієнтів
3. Ризики обслуговування	<u>Ризик затримки ремонту, перевантаження майстра, зниження оцінок, повторювані проблеми в категоріях робіт</u>	Service Health Radar	Дозволяють визначити, які фактори найбільше впливають на якість сервісу
4. Управлінські рішення	<u>Перерозподіл заявок, призначення майстра, контроль строків, аналіз проблемних категорій</u>	Dashboard, рекомендації Service Health Radar	<u>Допомогають адміністратору або менеджеру швидко реагувати на проблемні зони</u>

10

ДІАГРАМА КЛАСІВ ЛОГІКИ ОЦІНЮВАННЯ ЯКОСТІ ОБСЛУГОВУВАННЯ

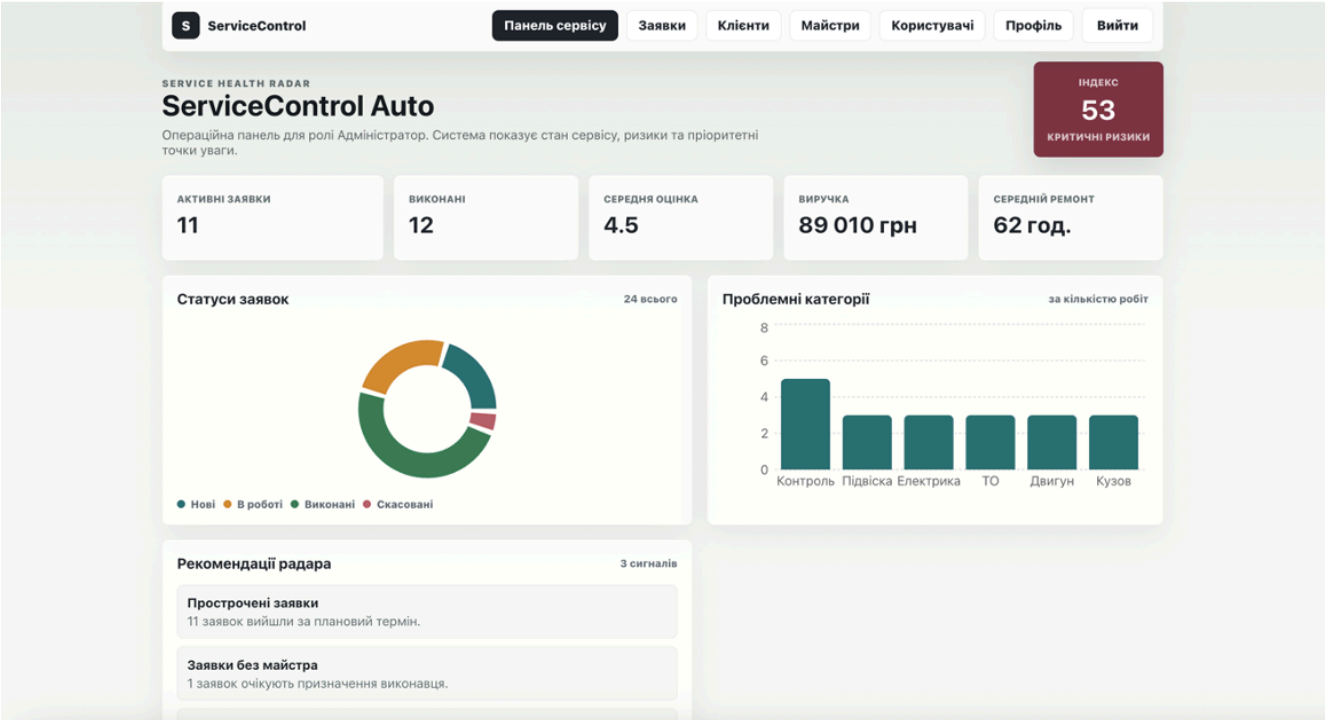


ЕКРАННІ ФОРМИ



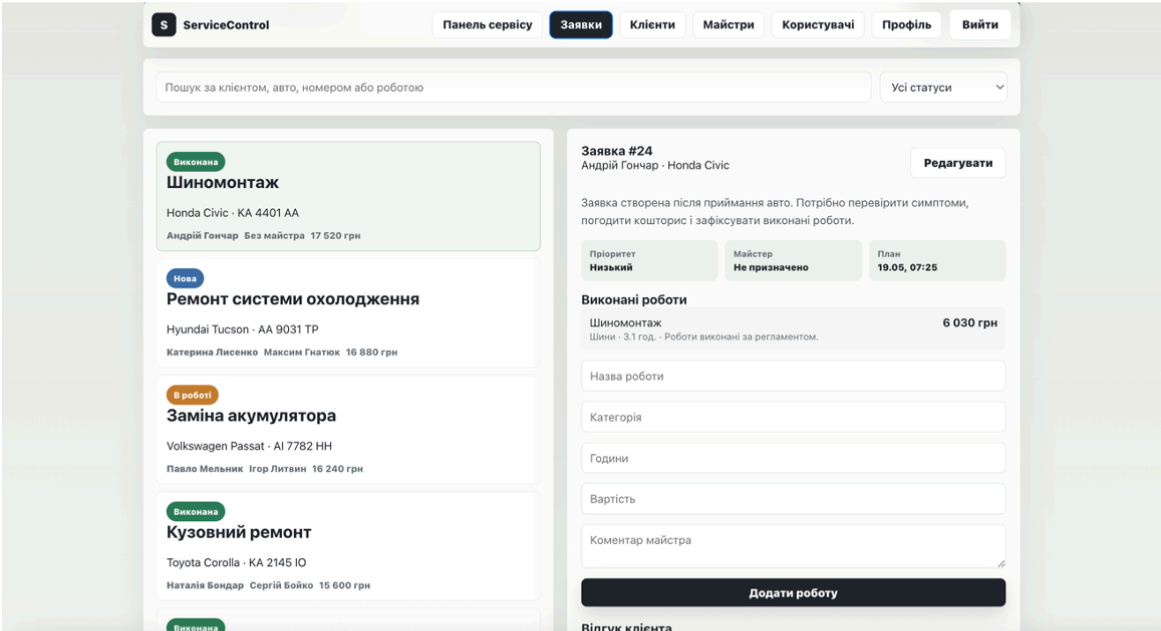
Сторінка реєстрації

ЕКРАННІ ФОРМИ



Головна сторінка адміністратора з аналітичними показниками

ЕКРАННІ ФОРМИ



Сторінка керування заявками на ремонт

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Сич Б.О., Хулік Б.О. Розробка платформи аналізу автосервісів для підвищення якості обслуговування. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.173-175.
2. Сич Б.О., Задонцев Ю.В. Особливості проектування інформаційної системи аналізу якості обслуговування в автосервісі. Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року, Київ, ДУІКТ. Збірник тез. (подано до друку).

16

ВИСНОВКИ

1. Проаналізовано процеси управління автосервісом і визначено проблеми, що впливають на якість обслуговування: прострочені заявки, нерівномірне навантаження майстрів, відсутність історії змін і слабка аналітика.
2. Досліджено аналоги AutoLeap, Shopmonkey, AutoFluent та ARI Auto Repair Software. Визначено, що вони мають базовий облік і звітність, але не дають зручної оцінки якості сервісу для малого/середнього автосервісу.
3. Сформовано вимоги до ServiceControl: рольовий доступ, облік клієнтів, авто, майстрів, заявок, виконаних робіт, відгуків, історії подій та аналітичних показників.
4. Обґрунтовано стек розробки: Java, Spring Boot, Spring Security, JWT, PostgreSQL, React, Vite, REST API, Docker та Git.
5. Спроектовано архітектуру, базу даних, процес обробки заявки, логіку оцінювання якості обслуговування та описано за допомогою UML-діаграм.
6. Реалізовано основні модулі ServiceControl: авторизацію, профіль, клієнтів, автомобілі, майстрів, користувачів, заявки, роботи, відгуки та історію подій.
7. Розроблено Service Health Radar, який аналізує прострочені заявки, завантаженість майстрів, заявки без виконавця, відгуки клієнтів і проблемні категорії робіт.
8. Проведено тестування системи. Підтверджено коректну роботу основних функцій, рольового доступу, Dashboard та аналітичного модуля Service Health Radar.

17

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Listing 1 — Entity: User.java

```
package com.example.servicecontrol.entity;

import
com.example.servicecontrol.entity.enums.UserRole;
import jakarta.persistence.*;
import lombok.*;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

import java.time.LocalDateTime;

@Entity
@Table(name = "users")
@Getter @Setter @NoArgsConstructor
@AllArgsConstructor @Builder
public class User {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length =
255)
    private String email;

    @Column(name = "password_hash", nullable = false)
    private String passwordHash;

    @Column(name = "full_name", length = 255)
    private String fullName;

    @Column(length = 40)
    private String phone;

    @Column(length = 120)
    private String position;

    @Column(name = "service_name", length = 180)
    private String serviceName;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 20)
    @Builder.Default
    private UserRole role = UserRole.MANAGER;

    @Builder.Default
    @Column(nullable = false)
    private boolean active = true;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "workspace_owner_id")
    private User workspaceOwner;

    @CreationTimestamp
```

```
@Column(name = "created_at", updatable = false)
private LocalDateTime createdAt;
```

```
@UpdateTimestamp
@Column(name = "updated_at")
private LocalDateTime updatedAt;
}
```

Listing 2 — Entity: Client.java

```
package com.example.servicecontrol.entity;
```

```
import jakarta.persistence.*;
import lombok.*;
import
org.hibernate.annotations.CreationTimestamp;
```

```
import java.time.LocalDateTime;
```

```
@Entity
@Table(name = "clients")
@Getter @Setter @NoArgsConstructor
@AllArgsConstructor @Builder
public class Client {
```

```
@Id
@GeneratedValue(strategy =
GenerationType.IDENTITY)
private Long id;
```

```
@ManyToOne(fetch = FetchType.LAZY, optional =
false)
```

```
@JoinColumn(name = "user_id", nullable = false)
private User user;
```

```
@Column(name = "full_name", nullable = false,
length = 180)
private String fullName;
```

```
@Column(nullable = false, length = 40)
private String phone;
```

```
@Column(length = 180)
private String email;
```

```
@Column(length = 800)
private String notes;
```

```
@CreationTimestamp
@Column(name = "created_at", updatable = false)
private LocalDateTime createdAt;
}
```

Listing 3 — Entity: Mechanic.java

```
package com.example.servicecontrol.entity;
```

```
import jakarta.persistence.*;
import lombok.*;
```

```

import java.math.BigDecimal;

@Entity
@Table(name = "mechanics")
@Getter @Setter @NoArgsConstructor
@AllArgsConstructor @Builder
public class Mechanic {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "account_user_id")
    private User accountUser;

    @Column(name = "full_name", nullable = false,
length = 180)
    private String fullName;

    @Column(nullable = false, length = 120)
    private String specialization;

    @Column(length = 40)
    private String phone;

    @Column(name = "hourly_rate", precision = 10, scale
= 2)
    private BigDecimal hourlyRate;

    @Builder.Default
    @Column(nullable = false)
    private boolean active = true;
}

```

Listing 4 — Entity: Vehicle.java
package com.example.servicecontrol.entity;

```

import jakarta.persistence.*;
import lombok.*;

```

```

@Entity
@Table(name = "vehicles")
@Getter @Setter @NoArgsConstructor
@AllArgsConstructor @Builder
public class Vehicle {

```

```

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @ManyToOne(fetch = FetchType.LAZY, optional =

```

```

false)
    @JoinColumn(name = "client_id", nullable = false)
    private Client client;

    @Column(nullable = false, length = 80)
    private String brand;

    @Column(nullable = false, length = 80)
    private String model;

    @Column(name = "plate_number", nullable = false,
length = 24)
    private String plateNumber;

    @Column(length = 40)
    private String vin;

    @Column(name = "production_year")
    private Integer productionYear;
}

```

Listing 5 — Entity: RepairOrder.java
package com.example.servicecontrol.entity;

```

import
com.example.servicecontrol.entity.enums.OrderPriority;
import
com.example.servicecontrol.entity.enums.OrderStatus;
import jakarta.persistence.*;
import lombok.*;

```

```

import java.math.BigDecimal;
import java.time.LocalDateTime;

```

```

@Entity
@Table(name = "repair_orders")
@Getter @Setter @NoArgsConstructor
@AllArgsConstructor @Builder
public class RepairOrder {

```

```

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "client_id", nullable = false)
    private Client client;

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "vehicle_id", nullable = false)
    private Vehicle vehicle;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "mechanic_id")
    private Mechanic mechanic;

```

```

@Enumerated(EnumType.STRING)
@Builder.Default
@Column(nullable = false, length = 24)
private OrderStatus status = OrderStatus.NEW;

@Enumerated(EnumType.STRING)
@Builder.Default
@Column(nullable = false, length = 16)
private OrderPriority priority =
OrderPriority.NORMAL;

@Column(name = "service_type", nullable = false,
length = 140)
private String serviceType;

@Column(nullable = false, length = 1200)
private String description;

@Column(name = "opened_at", updatable = false)
private LocalDateTime openedAt;

@Column(name = "planned_finish_at")
private LocalDateTime plannedFinishAt;

@Column(name = "completed_at")
private LocalDateTime completedAt;

@Column(name = "total_cost", precision = 12, scale =
2)
private BigDecimal totalCost;

@PrePersist
void prePersist() {
    if (openedAt == null) openedAt =
LocalDateTime.now();
}
}

```

Listing 6 — Entity: RepairWork.java
package com.example.servicecontrol.entity;

```

import jakarta.persistence.*;
import lombok.*;
import
org.hibernate.annotations.CreationTimestamp;

import java.math.BigDecimal;
import java.time.LocalDateTime;

@Entity
@Table(name = "repair_works")
@Getter @Setter @NoArgsConstructor
@AllArgsConstructor @Builder
public class RepairWork {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

```

```

@ManyToOne(fetch = FetchType.LAZY, optional =
false)
@JoinColumn(name = "order_id", nullable = false)
private RepairOrder order;

@Column(nullable = false, length = 160)
private String title;

@Column(nullable = false, length = 100)
private String category;

@Column(name = "hours_spent", precision = 8, scale
= 2)
private BigDecimal hoursSpent;

@Column(precision = 12, scale = 2)
private BigDecimal cost;

@Column(name = "mechanic_comment", length =
1000)
private String mechanicComment;

@CreationTimestamp
@Column(name = "created_at", updatable = false)
private LocalDateTime createdAt;
}

```

Listing 7 — Entity: OrderEvent.java
package com.example.servicecontrol.entity;

```

import jakarta.persistence.*;
import lombok.*;
import
org.hibernate.annotations.CreationTimestamp;

import java.time.LocalDateTime;

```

```

@Entity
@Table(name = "order_events")
@Getter @Setter @NoArgsConstructor
@AllArgsConstructor @Builder
public class OrderEvent {

```

```

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

```

```

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "order_id", nullable = false)
    private RepairOrder order;

```

```

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

```

```

    @Column(name = "event_type", nullable = false,
length = 80)
    private String eventType;

```

```

    @Column(nullable = false, length = 1000)

```

```

private String description;

@CreationTimestamp
@Column(name = "created_at", updatable = false)
private LocalDateTime createdAt;
}

```

Listing 8 — Entity: Feedback.java

```

package com.example.servicecontrol.entity;

```

```

import jakarta.persistence.*;
import lombok.*;
import org.hibernate.annotations.CreationTimestamp;

import java.time.LocalDateTime;

@Entity
@Table(name = "feedback")
@Getter @Setter @NoArgsConstructor
@AllArgsConstructor @Builder
public class Feedback {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @OneToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "order_id", nullable = false,
unique = true)
    private RepairOrder order;

    @Column(nullable = false)
    private Integer rating;

    @Column(length = 1000)
    private String comment;

    @CreationTimestamp
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;
}

```

Listing 9 — DTO: ServiceDtos.java

```

package com.example.servicecontrol.dto.autoservice;

```

```

import
com.example.servicecontrol.entity.enums.OrderPriority;
import
com.example.servicecontrol.entity.enums.OrderStatus;
import jakarta.validation.constraints.*;

```

```

import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.List;

```

```

public final class ServiceDtos {
    private ServiceDtos() {}
}

```

```

public record ClientRequest(
    @NotBlank String fullName,
    @NotBlank String phone,
    @Email String email,
    String notes
) {}

```

```

public record ClientResponse(
    Long id, String fullName, String phone,
    String email, String notes, LocalDateTime
createdAt
) {}

```

```

public record VehicleRequest(
    @NotNull Long clientId,
    @NotBlank String brand,
    @NotBlank String model,
    @NotBlank String plateNumber,
    String vin,
    Integer productionYear
) {}

```

```

public record VehicleResponse(
    Long id, Long clientId, String clientName,
    String brand, String model, String plateNumber,
    String vin, Integer productionYear
) {}

```

```

public record MechanicRequest(
    @NotBlank String fullName,
    @NotBlank String specialization,
    String phone, BigDecimal hourlyRate,
    Long accountUserId, Boolean active
) {}

```

```

public record MechanicResponse(
    Long id, String fullName, String specialization,
    String phone, BigDecimal hourlyRate, boolean
active,
    Long accountUserId, String accountEmail,
    Long activeOrders, Long completedOrders,
Double averageRating
) {}

```

```

public record RepairOrderRequest(
    @NotNull Long clientId,
    @NotNull Long vehicleId,
    Long mechanicId,
    OrderStatus status,
    OrderPriority priority,
    @NotBlank String serviceType,
    @NotBlank String description,
    LocalDateTime plannedFinishAt,
    LocalDateTime completedAt,
    BigDecimal totalCost
) {}

```

```

public record RepairWorkRequest(
    @NotBlank String title,
    @NotBlank String category,
    BigDecimal hoursSpent, BigDecimal cost,
    String mechanicComment
) {}

```

```

    ) {}

    public record RepairWorkResponse(
        Long id, String title, String category,
        BigDecimal hoursSpent, BigDecimal cost,
        String mechanicComment, LocalDateTime
        createdAt
    ) {}

    public record FeedbackRequest(@Min(1) @Max(5)
        Integer rating, String comment) {}

    public record FeedbackResponse(Long id, Integer
        rating, String comment, LocalDateTime createdAt) {}

    public record OrderEventResponse(
        Long id, String eventType, String description,
        String userName, LocalDateTime createdAt
    ) {}

    public record RepairOrderResponse(
        Long id, OrderStatus status, OrderPriority
        priority,
        String serviceType, String description,
        LocalDateTime openedAt, LocalDateTime
        plannedFinishAt,
        LocalDateTime completedAt, BigDecimal
        totalCost,
        ClientResponse client, VehicleResponse vehicle,
        MechanicResponse mechanic,
        List<RepairWorkResponse> works,
        FeedbackResponse feedback,
        List<OrderEventResponse> events
    ) {}

    public record AnalyticsSummaryResponse(
        long totalOrders, long activeOrders, long
        completedOrders,
        long cancelledOrders, long clients, long vehicles,
        long mechanics,
        BigDecimal revenue, BigDecimal
        averageRepairHours, Double averageRating
    ) {}

    public record HealthRiskResponse(String type, String
        title, String description, String severity) {}

    public record HealthRadarResponse(int score, String
        status, List<HealthRiskResponse> risks) {}

    public record StatusMetric(String status, long count)
    {}

    public record CategoryMetric(String category, long
        count, BigDecimal revenue) {}

    public record DashboardResponse(
        AnalyticsSummaryResponse summary,
        List<StatusMetric> statuses,
        List<CategoryMetric> categories,
        List<MechanicResponse> mechanics,
        List<RepairOrderResponse> recentOrders,
        HealthRadarResponse health

```

```

    ) {}
}
Listing 10 — Security: JwtTokenProvider.java
package com.example.servicecontrol.security;

import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import javax.crypto.SecretKey;
import java.nio.charset.StandardCharsets;
import java.util.Date;

@Component
public class JwtTokenProvider {

    @Value("${app.jwt.secret}")
    private String jwtSecret;

    @Value("${app.jwt.expiration-ms}")
    private long jwtExpirationMs;

    private SecretKey key() {
        return
        Keys.hmacShaKeyFor(jwtSecret.getBytes(StandardChar
        sets.UTF_8));
    }

    public String generateToken(String email) {
        Date now = new Date();
        return Jwts.builder()
            .subject(email)
            .issuedAt(now)
            .expiration(new Date(now.getTime() +
            jwtExpirationMs))
            .signWith(key())
            .compact();
    }

    public String getEmailFromToken(String token) {
        return Jwts.parser()
            .verifyWith(key()).build()
            .parseSignedClaims(token)
            .getPayload().getSubject();
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parser().verifyWith(key()).build().parseSignedClai
            ms(token);
            return true;
        } catch (JwtException | IllegalArgumentException
        e) {
            return false;
        }
    }
}
Listing 11 — Security: JwtAuthenticationFilter.java
package com.example.servicecontrol.security;

```

```

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtTokenProvider tokenProvider;
    private final UserDetailsServiceImpl userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain chain) throws ServletException, IOException {
        String header = request.getHeader("Authorization");
        if (header == null || !header.startsWith("Bearer ")) {
            chain.doFilter(request, response);
            return;
        }
        String token = header.substring(7);
        if (tokenProvider.validateToken(token)) {
            String email = tokenProvider.getEmailFromToken(token);
            UserDetails userDetails = userDetailsService.loadUserByUsername(email);
            UsernamePasswordAuthenticationToken auth = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
            auth.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

            SecurityContextHolder.getContext().setAuthentication(auth);
        }
        chain.doFilter(request, response);
    }
}

```

Listing 12 — Security: UserDetailsServiceImpl.java

```

package com.example.servicecontrol.security;

import com.example.servicecontrol.entity.User;
import com.example.servicecontrol.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.*;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {

    private final UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(email).orElseThrow(() -> new UsernameNotFoundException("User not found: " + email));
        if (!user.isActive()) {
            throw new UsernameNotFoundException("User disabled: " + email);
        }
        return org.springframework.security.core.userdetails.User.builder()
            .username(user.getEmail())
            .password(user.getPasswordHash())
            .authorities(List.of(new SimpleGrantedAuthority("ROLE_" + user.getRole().name())))
            .build();
    }
}

```

Listing 13 — Config: SecurityConfig.java

```

package com.example.servicecontrol.config;

import com.example.servicecontrol.security.JwtAuthenticationFilter;
import com.example.servicecontrol.security.UserDetailsServiceImpl;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

```

```

import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import
org.springframework.security.config.http.SessionCreationPolicy;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.cors.*;

```

```
import java.util.List;
```

```

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
@RequiredArgsConstructor
public class SecurityConfig {

```

```

    private final JwtAuthenticationFilter jwtAuthFilter;
    private final UserDetailsServiceImpl
userDetailsService;

```

```

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity
http) throws Exception {
        http
            .csrf(AbstractHttpConfigurer::disable)
            .cors(cors ->
cors.configurationSource(corsConfigurationSource()))
            .sessionManagement(s ->
s.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**", "/api-
docs/**",
                    "/swagger-ui/**", "/swagger-
ui.html").permitAll()
                .anyRequest().authenticated()
            )
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(jwtAuthFilter,

```

```

UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

```

```

    @Bean
    public CorsConfigurationSource
corsConfigurationSource() {
        CorsConfiguration config = new
CorsConfiguration();
        config.setAllowedOrigins(List.of(
            "http://localhost:3001",
            "http://127.0.0.1:3001",
            "http://localhost:5173", "http://127.0.0.1:5173"
        ));

```

```

        config.setAllowedMethods(List.of("GET", "POST", "PUT",
            "DELETE", "OPTIONS"));
        config.setAllowedHeaders(List.of("*"));
        config.setAllowCredentials(true);
        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", config);
        return source;
    }

```

```

    @Bean
    public DaoAuthenticationProvider
authenticationProvider() {
        DaoAuthenticationProvider provider = new
DaoAuthenticationProvider(userDetailsService);
        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }

```

```

    @Bean
    public PasswordEncoder passwordEncoder() { return
new BCryptPasswordEncoder(); }

```

```

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration
config) throws Exception {
        return config.getAuthenticationManager();
    }
}

```

Listing 14 — Service: AuthService.java
package com.example.servicecontrol.service;

```

import
com.example.servicecontrol.dto.request.LoginRequest;
import
com.example.servicecontrol.dto.request.RegisterRequest;
import
com.example.servicecontrol.dto.response.AuthResponse;
import com.example.servicecontrol.entity.User;
import
com.example.servicecontrol.exception.ConflictException
;
import
com.example.servicecontrol.repository.UserRepository;
import
com.example.servicecontrol.security.JwtTokenProvider;
import lombok.RequiredArgsConstructor;

```

```
import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;
```

```
@Service
@RequiredArgsConstructor
public class AuthService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtTokenProvider tokenProvider;
    private final AuthenticationManager authenticationManager;

    @Transactional
    public AuthResponse register(RegisterRequest request) {
        if
        (userRepository.existsByEmail(request.getEmail())) {
            throw new ConflictException("Email вже
            використується: " + request.getEmail());
        }
        User user = User.builder()
            .email(request.getEmail())

            .passwordHash(passwordEncoder.encode(request.getPassword()))
            .fullName(request.getFullName())
            .phone(request.getPhone())
            .position(request.getPosition())
            .serviceName(request.getServiceName())
            .build();
        userRepository.save(user);
        String token =
        tokenProvider.generateToken(user.getEmail());
        return new AuthResponse(token, user.getEmail(),
        user.getFullName(), user.getRole().name());
    }

    public AuthResponse login(LoginRequest request) {
        authenticationManager.authenticate(
            new
            UsernamePasswordAuthenticationToken(request.getEmail(),
            request.getPassword()));
        User user =
        userRepository.findByEmail(request.getEmail()).orElseThrow();
        String token =
        tokenProvider.generateToken(user.getEmail());
        return new AuthResponse(token, user.getEmail(),
        user.getFullName(), user.getRole().name());
    }
}
```

```
}
Listing 15 — Service: AutoService.java (fragment)
package com.example.servicecontrol.service;
```

```
import
com.example.servicecontrol.dto.autoservice.ServiceDtos.*;
import com.example.servicecontrol.entity.*;
import com.example.servicecontrol.entity.enums.*;
import com.example.servicecontrol.exception.*;
import com.example.servicecontrol.repository.*;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.Duration;
import java.util.*;
import java.util.stream.Collectors;
```

```
@Service
@RequiredArgsConstructor
public class AutoService {

    private final ClientRepository clientRepository;
    private final VehicleRepository vehicleRepository;
    private final MechanicRepository mechanicRepository;
    private final RepairOrderRepository orderRepository;
    private final RepairWorkRepository workRepository;
    private final FeedbackRepository feedbackRepository;
    private final OrderEventRepository orderEventRepository;
    private final UserRepository userRepository;

    @Transactional(readOnly = true)
    public List<RepairOrderResponse> orders(User user)
    {
        User workspace = scope(user);
        return visibleOrders(user).stream().map(order ->
        toOrder(workspace, order)).toList();
    }

    @Transactional
    public RepairOrderResponse saveOrder(User user,
    RepairOrderRequest request, Long id) {
        requireOfficeRole(user);
        User workspace = scope(user);
        Client client = client(workspace, request.clientId());
        Vehicle vehicle = vehicle(workspace,
        request.vehicleId());
        if (!Objects.equals(vehicle.getClient().getId(),
        client.getId())) {
            throw new
            IllegalArgumentException("Автомобіль не належить
            обраному клієнту");
        }
        RepairOrder order = id == null
        ?
        RepairOrder.builder().user(workspace).build()
    }
}
```

```

        : findOrder(workspace, id);
        order.setClient(client);
        order.setVehicle(vehicle);
        order.setMechanic(request.mechanicId() == null ?
null : mechanic(workspace, request.mechanicId()));
        order.setStatus(request.status() == null ?
OrderStatus.NEW : request.status());
        order.setPriority(request.priority() == null ?
OrderPriority.NORMAL : request.priority());
        order.setServiceType(request.serviceType());
        order.setDescription(request.description());

        order.setPlannedFinishAt(request.plannedFinishAt());
        order.setCompletedAt(request.completedAt());
        order.setTotalCost(request.totalCost());
        RepairOrder saved = orderRepository.save(order);
        logEvent(saved, user, id == null ?
"ORDER_CREATED" : "ORDER_UPDATED",
        id == null ? "Створено заявку" : "Оновлено
заявку");
        return toOrder(workspace, saved);
    }

    @Transactional
    public RepairWorkResponse addWork(User user,
Long orderId, RepairWorkRequest request) {
        User workspace = scope(user);
        RepairOrder order = findOrder(workspace, orderId);
        ensureVisibleOrder(user, order);
        RepairWork work = RepairWork.builder()
            .user(workspace).order(order)

        .title(request.title()).category(request.category())

        .hoursSpent(request.hoursSpent()).cost(request.cost())

        .mechanicComment(request.mechanicComment())
            .build();
        RepairWork saved = workRepository.save(work);
        logEvent(order, user, "WORK_ADDED", "Додано
роботу: " + saved.getTitle());
        return toWork(saved);
    }

    @Transactional(readOnly = true)
    public DashboardResponse dashboard(User user) {
        User workspace = scope(user);
        List<RepairOrder> orders = visibleOrders(user);
        List<RepairWork> works = orders.stream()
            .flatMap(o ->
workRepository.findByOrderOrderByIdAsc(o).stream()).
toList();
        List<Feedback> feedback = orders.stream()

        .map(feedbackRepository::findByOrder).flatMap(Option
al::stream).toList();

        long active = orders.stream().filter(o ->
o.getStatus() == OrderStatus.NEW ||
o.getStatus() == OrderStatus.IN_PROGRESS).count();
        long done = orders.stream().filter(o -> o.getStatus()
== OrderStatus.DONE).count();
        long cancelled = orders.stream().filter(o ->

```

```

o.getStatus() == OrderStatus.CANCELLED).count();
        BigDecimal revenue =
works.stream().map(RepairWork::getCost)

        .filter(Objects::nonNull).reduce(BigDecimal.ZERO,
BigDecimal::add);
        Double avgRating =
feedback.stream().map(Feedback::getRating)

        .filter(Objects::nonNull).mapToInt(Integer::intValue).av
erage().orElse(0);

        // ... aggregation, radar and mapping omitted for
brevity
        return new DashboardResponse(/* ... */);
    }

    private User scope(User user) {
        return user.getWorkspaceOwner() == null ? user :
user.getWorkspaceOwner();
    }

    private void requireOfficeRole(User user) {
        if (user.getRole() == UserRole.MECHANIC) {
            throw new IllegalArgumentException("Майстер
не має доступу до цієї дії");
        }
    }

    private void logEvent(RepairOrder order, User actor,
String type, String description) {
        orderEventRepository.save(OrderEvent.builder()
            .order(order).user(actor)
            .eventType(type).description(description)
            .build());
    }
}

```

Listing 16 — Controller: AuthController.java
package com.example.servicecontrol.controller;

```

import
com.example.servicecontrol.dto.request.LoginRequest;
import
com.example.servicecontrol.dto.request.RegisterRequest;
import
com.example.servicecontrol.dto.response.AuthResponse;
import com.example.servicecontrol.service.AuthService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

```

```

@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
@Tag(name = "Аутентифікація")
public class AuthController {

    private final AuthService authService;

    @PostMapping("/register")

```

```

    @ResponseStatus(HttpStatus.CREATED)
    @Operation(summary = "Реєстрація нового користувача")
    public AuthResponse register(@Valid @RequestBody RegisterRequest request) {
        return authService.register(request);
    }

    @PostMapping("/login")
    @Operation(summary = "Вхід у систему")
    public AuthResponse login(@Valid @RequestBody LoginRequest request) {
        return authService.login(request);
    }
}

```

Listing 17 — Controller: RepairOrderController.java
package com.example.servicecontrol.controller;

```

import
com.example.servicecontrol.dto.autoservice.ServiceDtos.*;
import com.example.servicecontrol.entity.User;
import com.example.servicecontrol.service.AutoService;
import com.example.servicecontrol.service.CurrentUserService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/orders")
@RequiredArgsConstructor
public class RepairOrderController {

    private final CurrentUserService currentUserService;
    private final AutoService autoService;

    @GetMapping
    public List<RepairOrderResponse> list(Authentication authentication) {
        return autoService.orders(user(authentication));
    }

    @GetMapping("/{id}")
    public RepairOrderResponse get(Authentication authentication, @PathVariable Long id) {
        return autoService.order(user(authentication), id);
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public RepairOrderResponse create(Authentication authentication,
                                      @Valid @RequestBody RepairOrderRequest request) {
        return autoService.saveOrder(user(authentication), request, null);
    }
}

```

```

    @PutMapping("/{id}")
    public RepairOrderResponse update(Authentication authentication,
                                      @PathVariable Long id,
                                      @Valid @RequestBody RepairOrderRequest request) {
        return autoService.saveOrder(user(authentication), request, id);
    }

    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void delete(Authentication authentication, @PathVariable Long id) {
        autoService.deleteOrder(user(authentication), id);
    }

    @PostMapping("/{id}/works")
    @ResponseStatus(HttpStatus.CREATED)
    public RepairWorkResponse addWork(Authentication authentication,
                                      @PathVariable Long id,
                                      @Valid @RequestBody RepairWorkRequest request) {
        return autoService.addWork(user(authentication), id, request);
    }

    @DeleteMapping("/works/{workId}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteWork(Authentication authentication, @PathVariable Long workId) {
        autoService.deleteWork(user(authentication), workId);
    }

    @PostMapping("/{id}/feedback")
    public FeedbackResponse feedback(Authentication authentication,
                                      @PathVariable Long id,
                                      @Valid @RequestBody FeedbackRequest request) {
        return autoService.saveFeedback(user(authentication), id, request);
    }

    private User user(Authentication authentication) {
        return currentUserService.get(authentication);
    }
}

```

Listing 18 — Controller: AnalyticsController.java
package com.example.servicecontrol.controller;

```

import
com.example.servicecontrol.dto.autoservice.ServiceDtos.DashboardResponse;
import com.example.servicecontrol.service.AutoService;
import com.example.servicecontrol.service.CurrentUserService;
import lombok.RequiredArgsConstructor;
import

```

```

org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/analytics")
@RequiredArgsConstructor
public class AnalyticsController {

    private final CurrentUserService currentUserService;
    private final AutoService autoService;

    @GetMapping("/dashboard")
    public DashboardResponse dashboard(Authentication
authentication) {
        return
autoService.dashboard(currentUserService.get(authentic
ation));
    }
}

```

Listing 19 — Exception: GlobalExceptionHandler.java
package com.example.servicecontrol.exception;

```

import org.springframework.http.*;
import
org.springframework.security.access.AccessDeniedExce
ption;
import
org.springframework.security.authentication.BadCredent
ialsException;
import
org.springframework.security.core.userdetails.Username
NotFoundException;
import org.springframework.validation.FieldError;
import
org.springframework.web.bind.MethodArgumentNotVal
idException;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.Map;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ProblemDetail
handleNotFound(ResourceNotFoundException ex) {
        return
ProblemDetail.forStatusAndDetail(HttpStatus.NOT_FO
UND, ex.getMessage());
    }

    @ExceptionHandler(ConflictException.class)
    public ProblemDetail
handleConflict(ConflictException ex) {
        return
ProblemDetail.forStatusAndDetail(HttpStatus.CONFLIC
T, ex.getMessage());
    }

    @ExceptionHandler({BadCredentialsException.class,
UsernameNotFoundException.class})

```

```

    public ProblemDetail
handleUnauthorized(RuntimeException ex) {
        return
ProblemDetail.forStatusAndDetail(HttpStatus.UNAUTH
ORIZED, "Невірний email або пароль");
    }

    @ExceptionHandler(AccessDeniedException.class)
    public ProblemDetail
handleAccessDenied(AccessDeniedException ex) {
        return
ProblemDetail.forStatusAndDetail(HttpStatus.FORBID
DEN, "Недостатньо прав для цієї дії");
    }

    @ExceptionHandler(MethodArgumentNotValidExceptio
n.class)
    public ProblemDetail
handleValidation(MethodArgumentNotValidException
ex) {
        Map<String, String> errors = new HashMap<>();
        for (FieldError err :
ex.getBindingResult().getFieldErrors()) {
            errors.put(err.getField(),
err.getDefaultMessage());
        }
        ProblemDetail pd =
ProblemDetail.forStatusAndDetail(HttpStatus.BAD_RE
QUEST, "Помилка валідації");
        pd.setProperty("errors", errors);
        return pd;
    }

    @ExceptionHandler(IllegalArgumentException.class)
    public ProblemDetail
handleBadRequest(IllegalArgumentException ex) {
        return
ProblemDetail.forStatusAndDetail(HttpStatus.BAD_RE
QUEST, ex.getMessage());
    }

    @ExceptionHandler(Exception.class)
    public ProblemDetail handleGeneral(Exception ex) {
        return
ProblemDetail.forStatusAndDetail(HttpStatus.INTERN
AL_SERVER_ERROR, "Внутрішня помилка
сервера");
    }
}

```