Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Центр післядипломної освіти
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту

(тема)

Виконав:
студент 4 курсу, групи ПЗПІп-22-1

Оболенцев Д.В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. Колесников Д.О.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Центр післядипломної освіти
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Оболенцеву Денису Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту
Затверджена наказом по університету від 17.06. 2024р. № 588 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 19.07.2024
3. Вихідні дані до роботи Розробити програмну систему у вигляді веб-застосунку, котрий дозволить зберігати та опрацьовувати дані підприємства з ремонту та технічного обслуговування автотранспорту, мовою програмування JavaScript, використовуючи для зберігання даних СКБД MySQL.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	26.05.2024	*виконано*
2	Створення специфікації ПЗ	01.06.2024	*виконано*
3	Проектування ПЗ	15.06.2024	*виконано*
4	Розробка ПЗ	18.06.2024	*виконано*
5	Тестування ПЗ	20.06.2024	*виконано*
6	Оформлення пояснювальної записки	05.07.2024	*виконано*
7	Підготовка презентації та доповіді	10.07.2024	*виконано*
8	Попередній захист	18.07.2024	*виконано*
9	Нормоконтроль, рецензування	18.07.2024	*виконано*
10	Здача роботи у електронний архів	18.07.2024	*виконано*
11	Допуск до захисту у зав. кафедри	19.07.2024	*виконано*

Дата видачі завдання 06 травня 2024р.

Студент (ка) _____ Оболенцев Д.В.
(підпис)

Керівник роботи _____ доц. Колесников Д.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 68 стор., 22 рис., 4 табл., 18 джерел.

ВЕБ-ЗАСТОСУНОК, ЗАПИС, ЗВІТ, КЛІЄНТ, ПОСЛУГИ, РЕЛЯЦІЙНА БАЗА ДАНИХ, РЕМОНТНА РОБОТА, СТАТИСТИКА, ТРАНСПОРНИЙ ЗАСІБ, ФАХІВЕЦЬ, BACK-END, FRONT-END, MYSQL, NODE.JS, REACT.JS.

Об’єкт розробки – програмна система для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту.

Мета розробки – створення програмної системи для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту, що дозволяє зберігати та обробляти дані про клієнтів, транспортні засоби, що перебувають в ремонті чи були відремонтовані та проведені ремонтні роботи.

Метод рішення – середовище розробки Visual Studio Code, мова програмування JavaScript, система управління базою даних MySQL.

У результаті розробки створено програмну систему для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту, котра дозволяє створювати обліковий кабінет клієнта, формувати запис на прийом до майстра з ремонту автотранспорту, зберігати дані про працівників, клієнтів, їх транспортні засоби, проведені ремонтні роботи, та здійснювати розподіл задач між працівниками та отримувати статистику про поточну діяльність підприємства та фахівців.

WEB APPLICATION, MAKE AN APPOINTMENT, REPORT, CUSTOMER, SERVICES, RELATIONAL DATABASE, REPAIR WORK, STATISTICS, VEHICLE, SPECIALIST, BACK-END, FRONT-END, MYSQL, NODE.JS, REACT.JS.

The object of development is a software system for management and accounting for the work of a vehicle repair and maintenance company.

The goal of development is a creation a software system for management and accounting for the work of a vehicle repair and maintenance company, which allows storing and processing data about customers, vehicles that are repaired or have been repaired and completed repair works.

The solution method is the Visual Studio Code development environment, the JavaScript programming language, and the MySQL database management system.

As a result of the development, a software system was created for the management and accounting of the work of a motor vehicle repair and maintenance company, which allows to create a customer account, make an appointment to a vehicle repair master, store data about employees, customers, their vehicles, performed repair work, and distribute tasks among employees and receive statistics on the current activity of the company and specialists.

Я, Оболенцев Денис Васильович, студент гр. ПЗПІп-22-1, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 7
1 Аналіз предметної галузі 8
1.1 Аналіз предметної галузі 8
1.2 Виявлення та вирішення проблем 10
1.3 Постановка задачі 16
2 Формування вимог до програмної системи 18
3 Архітектура та проєктування програмного забезпечення 21
3.1 UML проєктування ПЗ 21
3.2 Проєктування архітектури ПЗ 25
3.3 Проєктування структури зберігання даних 27
3.4 Приклади найцікавіших алгоритмів та методів 30
3.5 Створення UI / UX дизайну системи 31
4 Опис прийнятих програмних рішень 34
4.1 Опис реалізації серверної частини 34
4.2 Опис реалізації клієнтської частини 40
5 Тестуваннння розробленого програмного забезпечення 47
Висновки 51
Перелік джерел посилання 52
Додаток А 54
Додаток Б 55
Додаток В 63

ВСТУП

У сучасну цифрову епоху для кожної компанії вкрай важливо використовувати сучасні інформаційні технології, що зумовлено потребою у постійному вдосконаленні своєї діяльності, збільшенні задоволеності клієнтів та ефективній комунікації зі споживачем в режимі реального часу. Використання веб-застосунків для здійснення онлайн-запису на прийом до фахівців, застосування електронної черги, прозоре інформування про послуги та їх вартість посилює лояльність клієнтів до компанії.

Впровадження ефективної програмної системи може надати можливість підприємствам з ремонту та технічного обслуговування автотранспорту власні оптимізувати операційні процеси, зменшити кількість помилок, котрі можуть статися при ручному, не автоматизованому обліку клієнтів, автотранспорту та проведених ремонтних робіт. Крім того, облік у паперовому форматі відомостей про клієнтів, транспортні засоби та проведені ремонтні роботи засоби ускладнює проведення контролю та управління діяльністю підприємства з ремонту та технічного обслуговування автотранспорту. Автоматизуючи завдання з обліку клієнтів та ремонтних робіт та надаючи доступ до інформації в режимі реального часу клієнтам та фахівцям з ремонту, підприємство може підвищити свою конкурентоспроможність, підвищити лояльність клієнтів і знизити операційні витрати, що сприятиме збільшенню прибутковості діяльності у майбутньому.

Основною метою кваліфікаційної роботи є створення програмної системи для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту, що дозволяє зберігати та обробляти дані про клієнтів, транспортні засоби, що перебувають в ремонті та проведені роботи з технічного обслуговування.

Об’єкт дослідження: програмна система для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту.

Предмет дослідження: застосування сучасних технологій створення програмного забезпечення для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Аналіз предметної області діяльності підприємства з ремонту та технічного обслуговування автотранспорту дозволяє виявити проблемні питання, які необхідно вирішити в ході виконання роботи. Спочатку варто розглянути поняття «технічне обслуговування автотранспорту» та «ремонт». Для цього можна розглянути нормативно-правові акти. Так, наказ Міністерства транспорту України № 102 від 30 березня 1998 року «Про затвердження Положення про технічне обслуговування і ремонт дорожніх транспортних засобів автомобільного транспорту» надає визначення поняттю «ремонт» як комплекс операцій з відновлення справності чи працездатності. Відповідно технічне обслуговування автотранспорту – це комплекс операцій з підтримки працездатності або справності транспортного засобу [1].

Поняття підприємство з ремонту та технічного обслуговування автотранспорту визначено наказом Міністерства інфраструктури України № 550 від 26.07.2013 року «Про затвердження Правил експлуатації колісних транспортних засобів» як підприємство технічного сервісу – підприємство, яке забезпечує технічні операції і процеси з утримання транспортних засобів у безпечному технічному стані відповідно до технічних умов виробника та норм законодавства [2].

Метою технічного обслуговування та ремонту є підтримка транспортних засобів у технічно справному стані, забезпечення надійності експлуатації та безпеки руху. Технічне обслуговування та ремонт транспортних засобів виконується у відповідності до технологічного процесу у спеціально обладнаних приміщеннях, на робочих постах. Робочі пости – спеціальні ділянки оснащені необхідним обладнанням та засобами, мають площу необхідну для розміщення автомобіля і можуть включати одне або декілька робочих місць для фахівця з ремонту та технічного обслуговування [3]. Робочі пости поділяються за технологічним призначенням на універсальні, які дозволяють виконувати усі

операції, та спеціалізовані, де можна виконати лише одну чи декілька операцій з ремонту та технічного обслуговування.

Підприємства можуть використовувати два основних методи технічного обслуговування та ремонту автотранспорту – одиничний та потоковий методи.

Одиничний метод полягає у виконанні всіх робіт лише на одному універсальному посту та зазвичай використовується на невеликих автомайстернях. Для цього методу характерні велика тривалість часу обслуговування та низька продуктивність праці. Застосування потокового методу дозволяє виконувати завдання з ремонту та технічного обслуговування автотранспорту на декількох спеціалізованих постах, котрі разом утворюють потокову лінію. Для цього методу характерні висока продуктивність праці та висока якість обслуговування. Потоковий метод реалізується за умови дотримання графіку виконання робіт та чіткого розподілення завдань між виконавцями, фахівцями з ремонту та технічного обслуговування, адже у випадку порушення графіку виконання робіт на одному з постів виникають затримки, простої на інших постах, що порушує робочий процес та строки виконання робіт з ремонту та технічного обслуговування [4].

Виконання технічного обслуговування та ремонту транспортних засобів здійснюється фахівцями відповідної кваліфікації та спеціалізації відповідно до визначеного технологічного процесу. Технологічний процес ремонту та технічного обслуговування зазначається у технологічних картах, в яких описано увесь процес дій на транспортний засіб, де вказані у визначеній послідовності операції, кваліфікація виконавців, технологічне оснащення, норми часу, технічні умови та вказівки. Технологічні карти розподіляються на операційні та постові. Операційні карти містять перелік дій, що виконуються по агрегатам, вузлам, системам автомобіля. Постові карти містять перелік дій, що виконуються на конкретному робочому посту [3].

Отже, підприємство з ремонту та технічного обслуговування автотранспорту здійснює проведення технологічного обслуговування та ремонтних робіт транспортних засобів осіб, що звернулися до підприємства – клієнтів. Загальна структура підприємства складається з відділу роботи з клієнтами, ремонтного цеху,

бухгалтерія, відділ кадрів та офісу директора. Роботи з ремонту та технічного обслуговування виконуються кваліфікованими фахівцями, що мають достатній рівень знань та навичок, освітньої підготовки та стажу.

1.2 Виявлення та вирішення проблем

З метою розробки програмної системи необхідно провести аналіз існуючих рішень та методів. Це дозволяє виявити стан сучасних програмних систем, їх недоліки та переваги, що дозволить сформувати цілі та задачі для розробки власної програмної системи для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту.

Наразі на вітчизняному на закордонному ринку з’являються щорічно нові програмні системи, веб-застосунки для вирішення завдань з управління та обліку роботи. Кожне керівництво та власники підприємств з ремонту та технічного обслуговування автотранспорту, які розуміють важливість використання сучасних інформаційних технологій у бізнесі, замовляють розробку або купують готові програмні рішення для покращення обліку роботи, клієнтів свого підприємства.

Одним з провідних рішень в галузі ведення управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту є онлайн-платформа EasyWeek [5], що допомагає бізнесу в ефективному управлінні та автоматизації обліку клієнтів та завдань. Завдяки EasyWeek клієнти мають можливість переглянути усі послуги, які надає підприємство, та здійснити онлайн запис на прийом до фахівця. В свою чергу адміністратор підприємства може відстежувати та керувати записами та отримувати статистику з візитів, виконаних робіт.

Перевагами програмної системи EasyWeek є:

- доступ до додатку здійснюється за допомогою браузера;
- перегляд історії обслуговування для кожного клієнта;
- наявність зручного календаря для швидкого онлайн-запису;
- можливість отримати статистики та звіти з виконаних робіт;
- інтеграція з сервісами онлайн-оплати;

- можливість переглянути усі послуги, які надає підприємство;
- сповіщення клієнта за допомогою SMS та email [5].

До недоліків EasyWeek можна віднести:

- відсутність облікового запису клієнта;
- відсутність можливості збереження інформації про транспортні засоби;
- не передбачена можливість формування клієнтом акту виконаних робіт.

Інтерфейс програми зображено на рисунку 1.1.

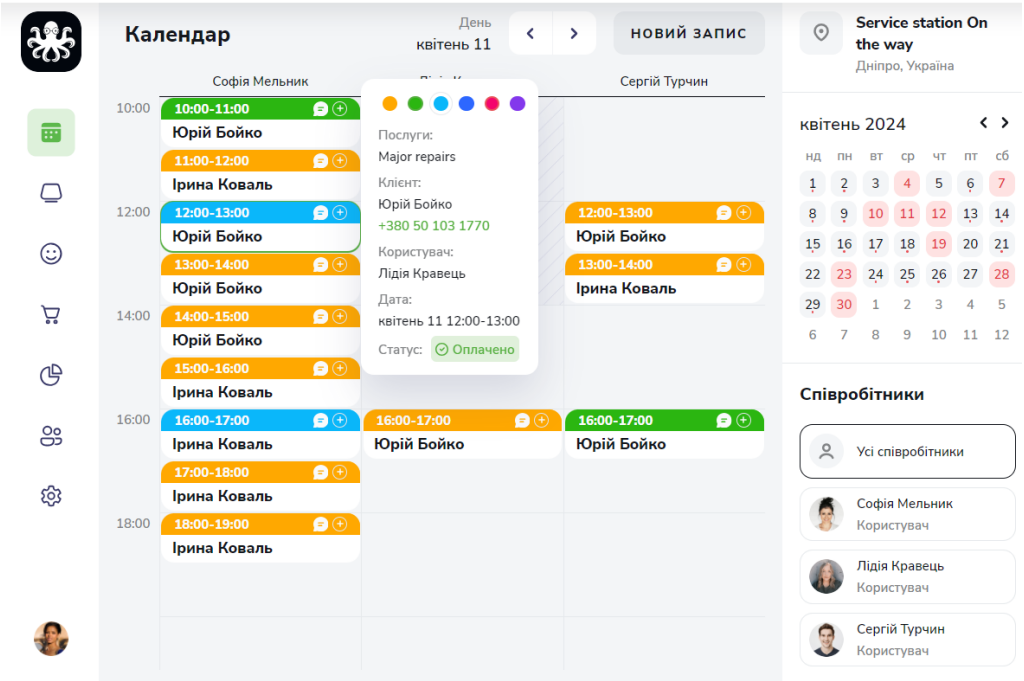


Рисунок 1.1 – Онлайн-платформа EasyWeek (за даними [5])

Ще одним програмним рішенням для забезпечення управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту є AutoRepair Cloud – хмарне програмне забезпечення, отримати доступ до якого можливо за допомогою мережі Інтернет [6].

До переваг програмної системи AutoRepair Cloud можна віднести:

- доступ до системи можливий як через браузер так і за допомогою додатку для мобільних пристроїв;
- онлайн-запис на прийом до фахівця;
- можливість переглянути хід виконання робіт з ремонту та технічного обслуговування;

- можливість сформувати акт виконаних робіт та роздрукувати чи надіслати на електронну пошту;
- інтеграція з сервісами онлайн-оплати;
- можливість отримати статистики та звіти з виконаних робіт;
- можливість призначити виконавця для виконання робіт з ремонту та технічного обслуговування;
- можливість перегляду історії обслуговування для кожного клієнта;
- можливість перегляду усіх послуг, що надає підприємство [6].

Серед недоліків можна зазначити відсутність локалізації та перевантажений великою кількістю елементів інтерфейс користувача. Інтерфейс програмної системи зображено га рисунку 1.2.

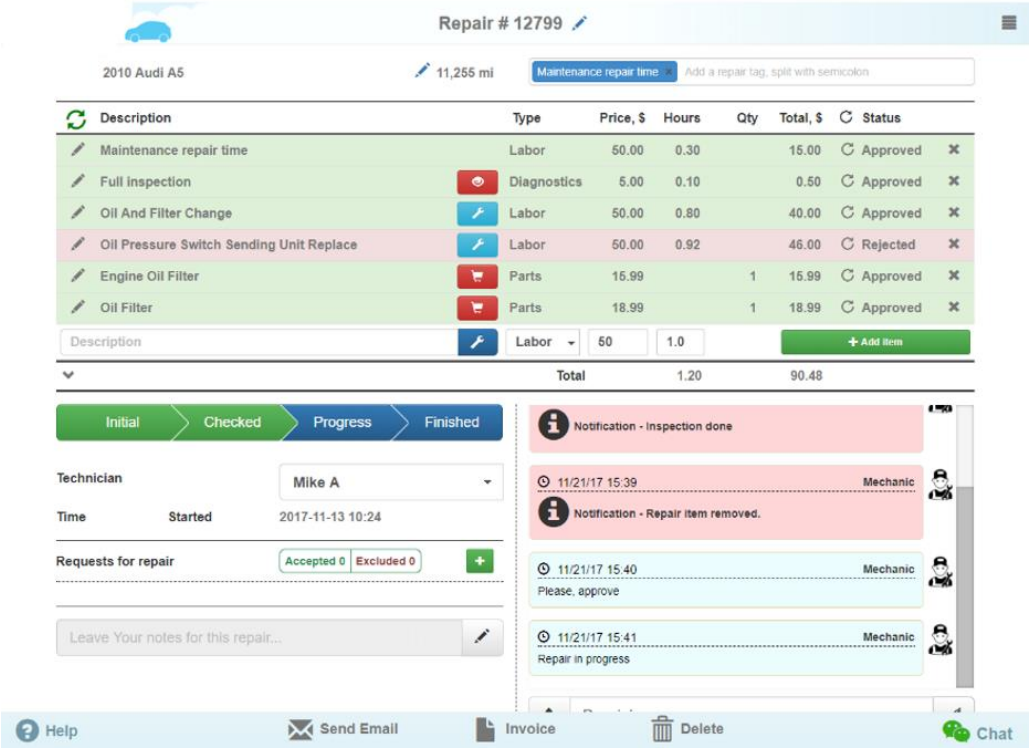


Рисунок 1.2 – Інтерфейс програмної системи AutoRepair Cloud (за даними [6])

Можливість обліку роботи та клієнтів підприємства з ремонту та технічного обслуговування автотранспорту забезпечує онлайн-платформа autobooking.com, що дозволяє власникам підприємств з ремонту та технічного обслуговування автотранспорту створити обліковий запис – сторінку з повним описом підприємства та послуг, які вони надають. Клієнт може знайти необхідний

автосервіс за розташуванням, необхідними послугами, відгуками та записатися онлайн до фахівця з ремонту та технічного обслуговування.

До переваг онлайн-платформи autobooking.com можна віднести:

- можливість моніторингу записів та обліку клієнтів та завантаженості постів;
- сповіщення клієнта за допомогою SMS та email;
- можливість перегляду списку послуг, що надає підприємство;
- можливість оцінити якість надання послуг та залишити відгук [7].

До недоліків онлайн-платформи autobooking.com можна віднести:

- відсутність можливості збереження інформації про транспортні засоби;
- не передбачена можливість формування клієнтом акту виконаних робіт.

На рисунку 1.3 зображено головну сторінку веб-порталу autobooking.com.

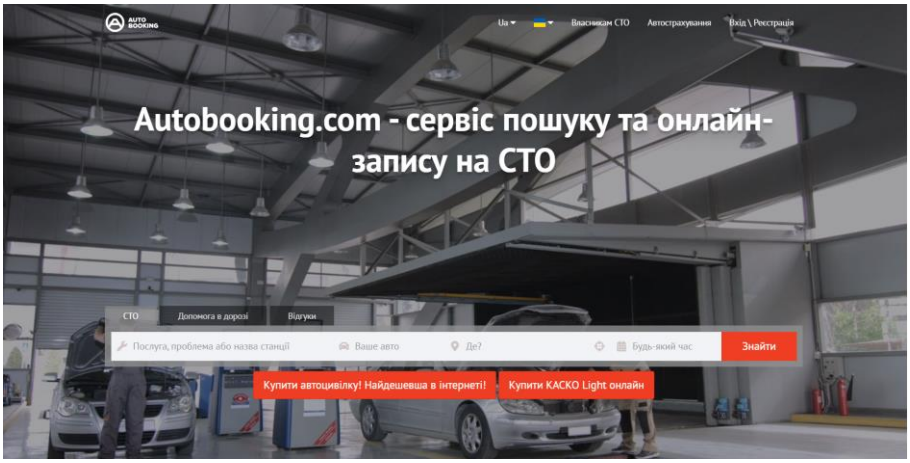


Рисунок 1.3 – Головна сторінка веб-порталу autobooking.com (за даними [7])

Після проведення аналізу існуючих програмних рішень можна зробити висновок, що програмна система для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту являє собою централізовану автоматизовану систему обліку клієнтів та ремонтних робіт, котра забезпечує точний та надійний облік всіх аспектів діяльності підприємства з технічного обслуговування та ремонту автотранспорту, включаючи інформацію про типи послуг, проведені ремонтні роботи, клієнтів та відремонтовані автотранспортні засоби, фахівців що здійснюють ремонт та технічне обслуговування. Система дозволяє централізований доступ до інформації, забезпечує точність та надійність даних, а також відповідність чинному законодавству.

Програмна система дозволяє забезпечити доступ до інформації про послуги підприємства для клієнтів, які зможуть заздалегідь розрахувати вартість робіт. Програмна надає можливість клієнтам здійснити онлайн-запис на прийом до фахівців з технічного обслуговування у обраний час та відстежувати історію проведених робіт для кожного транспортного засобу.

За прийом автотранспорту від клієнтів на ремонт відповідає менеджер адміністратор, також менеджер здійснює видачу відремонтованих автомобілів, контролює сплату рахунків за послуги, укладає договори на корпоративне технічне обслуговування автотранспорту. Здебільшого автомайстерні – підприємства малого та середнього бізнесу, що зумовлює оптимізацію кадрових ресурсів, так адміністратор – менеджер може виконувати функції співробітника кадрового підрозділу, а саме приймати на роботу, звільняти інженерів-фахівців з ремонту автотранспорту.

Отже, мета роботи менеджера – адміністратора полягає в організації та підтримці діяльності підприємства. Процес з організації та підтримки діяльності підприємства з ремонту та технічного обслуговування автотранспорту полягає в тому, що адміністратори виконують технічну роботу, а саме підтримують в актуальному стані інформацію про клієнтів, автомобілі передані в ремонт, особовий склад інженерів-фахівців з ремонту автотранспорту, про послуги, що надає підприємство, про проведені ремонтні роботи, формують акти виконаних робіт, укладають договори на надання послуг з ремонту та технічного обслуговування автотранспорту, відстежують статус виконання ремонтних робіт, здійснюють призначення фахівців на ремонтні роботи.

Інформаційні потреби адміністраторів підприємства з ремонту та технічного обслуговування автотранспорту:

- необхідна інформація про нових клієнтів, що потрапляє до відділу роботи з клієнтами у вигляді договору на надання послуг з ремонту та технічного обслуговування автотранспорту в електронному вигляді (формат .docx);

- необхідні назви моделей та брендів vin-коди авто, що надаються клієнтами для ремонту; це поступає в вигляді документів технічного паспорту транспортного засобу (паперовий вигляд);
- результати виконання ремонтних робіт поступають у вигляді актів проведених робіт в паперовому вигляді;
- пошук серед наявних клієнтів за обраним фільтром (прізвище, номер, тощо);
- пошук в базі даних збережених транспортних засобів за обраним фільтром (бренд, модель, реєстраційний номер тощо);
- пошук серед записів ремонтних робіт за обраним фільтром (дата виконання, вартість тощо);
- адміністратори повинні мати змогу формувати звіти про поточні результати діяльності працівників (кількість та вартість проведених ремонтних робіт) для надання керівництву та власникам у вигляді добре структурованого звіту з поточної діяльності підприємства.

Головна проблема, котру повинна вирішувати програмна система (ПС) – автоматизація процесів обліку клієнтів та ремонтних робіт, включаючи їх вартість, терміни виконання. Таким чином забезпечується зменшення ручної праці, підвищення точності і надійності даних, та зростання ефективності моніторингу та управління бізнес процесами.

Отже, основним завданням, яке вирішує програмна система, є збереження відомостей про клієнтів, транспортні засоби та проведені ремонтні роботи. Загальна контекстна діаграма програмної системи наведена на рисунку 1.4.

Рисунок 1.4 – Контекстна діаграма програмної системи (рисунок виконаний самостійно)

1.3 Постановка задачі

Необхідно спроектувати програмну систему, яка здійснює обробку та збереження інформації стосовно послуг, що надаються підприємством з ремонту та технічного обслуговування автотранспорту, працівників, клієнтів, їх транспортних засобів, проведених робіт з ремонту та технічного обслуговування. Програмна система повинна забезпечувати наступні функції:

- авторизація користувачів;
- надання доступу до функціоналу програми з урахуванням статусу користувача (клієнт, фахівець з технічного обслуговування, адміністратор);
- реєстрація нового клієнта;
- відображення облікового профілю користувача;
- внесення змін до облікових даних користувача;
- зміна та відновлення паролю для входу в систему;
- видалення облікового запису;
- система повинна відображати дані про основні поняття предметної області (ПО): працівників, автомобілі, послуги підприємства з ремонту та технічного обслуговування автотранспорту та інформацію про автозапчастини, інформацію про клієнтів - власників автотранспорту.
- система повинна підтримувати можливість визначення загальної кількості працівників; загальної кількості проведених ремонтних робіт фахівцем;
- система повинна підтримувати можливість визначення кількості транспортних засобів, що перебувають в ремонті;
- система повинна підтримувати сортування працівників за спеціальністю, автомобілів за назвою, ПІБ клієнтів, ремонтні роботи за датою проведення, за вартістю;
- необхідно передбачити можливість здійснювати пошук інформації про клієнта за його повним або частковим ПІБ; про автомобіль за його реєстраційним номером або vin-кодом;
- треба забезпечити можливість здійснювати фільтрацію інформації про ремонтні роботи за транспортним засобом, працівником, датою виконання.

- система повинна підтримувати додавання, редагування даних про автомобілі, працівників, клієнтів, послуги;
- система повинна підтримувати можливості видалення інформації про автомобілі, працівників, клієнтів, послуги з підтримкою режиму підтвердження користувачем видалення інформації про поточний об’єкт.

Для підтримки діяльності менеджера необхідно забезпечити можливість отримання інформації щодо:

- ТОП 5 найпоширеніших несправностей в автомобілях (назва несправності, марка авто, кількість звернень, загальна вартість виконаних робіт);
- визначення кращого співробітника (ПІБ, кількість виконаних завдань, загальна вартість виконаних робіт);
- клієнта, який найчастіше відвідує СТО (ПІБ, кількість звернень, загальна сума сплати за послуги);

Також система повинна підтримувати підготовку та друк наступних звітів:

- акт виконаних робіт (ПІБ виконавця, ПІБ клієнта, реєстраційний номер авто, назва виконаної роботи, ціна заміненої автозапчастини, кількість, вартість виконаної роботи, загальна сума до сплати);
- картка технічного стану авто (реєстраційний номер авто, марка авто (VIN-номер), ПІБ клієнта, дата виконання та назви проведених робіт, їх загальна кількість та вартість).

Програмна система має реалізовувати задачу з підбору виконавця ремонтної роботи з урахуванням дати проведення роботи, її вартості, зайнятості фахівця та його досвіду.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Програмна система повинна бути реалізована у вигляді веб-додатку, доступ до системи має здійснюватися за допомогою браузера та працювати з будь-якою операційною системою як на десктопних так і мобільних пристроях. Веб-додаток – це прикладне програмне забезпечення, логіка якого розподілена між сервером та клієнтом [8].

Інтерфейс користувача має бути зручним та інтуїтивно зрозумілим для користувача з будь-яким рівнем навичок користування інформаційними системами. Веб-додаток не повинен бути перевантаженим великою кількістю зайвих елементів.

Програмна система повинна вирішувати задачі, що наведено у розділі 1.3. Програмна система передбачає розподіл користувачів за наступними ролями: клієнт-користувач, фахівець з ремонту та технічного обслуговування автотранспорту та адміністратор програмної системи. Для кожного користувача створюється відповідний особистий кабінет. Крім того, у відповідності до ролі користувача, його надаються різні права доступу та функціональні можливості програмної системи.

Програмна система має складатися з серверної та клієнтської частин. Серверна частина отримує запити та дані з клієнтської частини. Дані з серверної частини повинні завантажуватися та відображатися на клієнтській частині. Крім того, серверна частина взаємодіє з базою даних та забезпечує читання, запис, редагування та видалення даних.

Програмна система має бути розроблена мовою програмування JavaScript. Для проектування серверної частини (backend) повинна бути використана програмна платформа Node.js.

Серверна частина повинна виконувати наступні задачі:

- реєстрація та авторизація клієнта;
- забезпечення захищеного зберігання паролів необхідних для входу в систему шляхом хеш-шифрування паролів;
- авторизація фахівців та адміністратора;

- розрахунок вартості виконаних робіт;
- створення запису на прийом до фахівця з ремонту;
- формування статистики з поточної діяльності підприємства;
- формування звіту про виконані ремонтні роботи;
- формування звіту картки технічного стану транспортного засобу;
- додавання, редагування, видалення інформації про усі сутності бази даних;
- виконання валідації введених даних перед збереженням в базі даних;
- надавати та обмежувати доступ до даних та функціоналу програмної системи користувачам у відповідності до їх ролей та повноважень;
- обробка запиту з підбору виконавця ремонтної роботи з урахуванням зайнятості та досвіду виконавця;

З метою забезпечення цілісності та надійності зберігання даних необхідно використати реляційну базу даних, що застосовується для зберігання та організації доступу до даних, котрі взаємопов’язані між собою.

В основі реляційних баз даних лежить реляційна модель – наочне подання інформації у вигляді взаємопов’язаних таблиць. Кожен рядок у таблиці має унікальний ідентифікатор – ключ, а стовпці таблиць містять атрибути даних, що дозволяє встановлювати зв'язки між елементами даних. Реляційна база даних – це набір відношень, імена яких збігаються з іменами схем відношень у схемі бази даних [9]. Саме тому, реляційна модель бази даних є найбільш ефективною для вирішення задачі з розробки програної системи для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту. Для зберігання даних програмної системи повинна бути використана реляційна база даних MySQL, що забезпечить чітку структуровану модель для кожної сутності та можливість отримувати за допомогою запитів необхідні дані.

Клієнтська частина забезпечує взаємодію з користувачем за допомогою зручного інтерфейсу, потім надсилає дані до серверної частини, де здійснюється їх опрацювання, після чого серверна частина повертає опрацьований результат та забезпечує роботу клієнта з даними, що зберігаються в базі даних.

Для написання клієнтської частини (frontend), має бути використана бібліотека React.js. Клієнтська частина повинна забезпечувати наступні задачі:

- надання зручного та інтуїтивно зрозумілого користувацького інтерфейсу;
- підтримувати роботу на будь-якому типі пристрою клієнта (десктопна чи мобільна версія), мати адаптивний дизайн;
- завантаження сформованих звітів на пристрій користувача;
- перегляд статистики з поточної діяльності підприємства з ремонту та технічного обслуговування автотранспорту;
- з метою уникнення помилок у роботі програмної системи виконувати валідацію введених користувачем даних перед відправкою на сервер;

В предметній області з ремонту та технічного обслуговування автотранспорту існує рід обмежень, які можна віднести до обмежень цілісності стосовно зав’язків:

- одному клієнту можуть належать декілька транспортних засобів, але один транспортний засіб може належати лише одному клієнту (з яким укладено угоду на надання ремонтних послуг);
- один фахівець може надавати декілька послуг, але одна ремонтна робота виконується одноособово (одним фахівцем);
- до кожного автомобіля може бути виконано декілька ремонтних робіт, але одна ремонтна робота виконується лише для одного автомобіля;
- кожна послуга може бути надана під час декількох ремонтних робіт, протягом одної ремонтної роботи може бути надано декілька послуг.

З урахуванням законодавчих вимог, пов'язаних з діяльністю підприємства з ремонту та технічного обслуговування автотранспорту необхідно дотримуватися наступних обмежень:

- облік ремонтних робіт та зберігання і обробка персональних даних проводиться згідно чинного законодавства;
- використовувати для розробки програмної системи ліцензоване програмне забезпечення.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Уніфікована мова моделювання (Unified Modeling Language, UML) застосовується з метою розробки детального плану програмної системи, що відображає концептуальні елементи (системні функції, бізнес-процеси) та особливості її реалізації наприклад схеми баз даних, діаграми розгортання. Наразі уніфікована мова моделювання широко використовується розробниками програмного забезпечення та стала загальноприйнятим стандартом для графічного опису програмного забезпечення. Уніфікована мова моделювання – це графічна мова орієнтована на представлення моделей програмних систем у графічній формі, застосовується для візуалізації, специфікації та документування усіх артефактів, які створюються під час програмних систем.

Діаграми UML являють собою графічне зображення елементів системи у формі зв’язаного графа з вершинами – сутностями та ребрами – відношеннями.

Одним з різновидів діаграм UML є діаграма варіантів використання або діаграма прецедентів (Use case diagram) – це узагальнене представлення функціонального призначення програмної системи, на якому зображено варіанти використання (прецеденти) та діючі особи (актори) – користувачі програмної системи та відношення між ними. Варіанти використання системи дозволяють визначити можливості та межі програмної системи [10].

Для роботи з програмною системою користувач може бути не авторизованим в такому разі йому буде доступний обмежений функціонал системи, а саме: пошук послуг підприємства та розрахунок вартості ремонту авто та реєстрація нового клієнта. Якщо користувач авторизований, то йому доступний повний функціонал програмної системи в залежності від ролі користувача.

Зареєстрований клієнт має можливість редагувати дані свого облікового профілю та у разі необхідності видалити його, також клієнт може додавати, редагувати, видаляти дані про свої транспортні засоби. Крім того, клієнт може створювати, видаляти, та змінювати запис на прийом до фахівця з технічного обслуговування та ремонту. Клієнт має можливість сформувати та роздрукувати

або завантажити звіт про виконану роботу чи звіт про стан свого транспортного засобу. Також авторизований клієнт може оцінити якість наданих послуг та залишити свій відгук.

Фахівець з ремонту технічного обслуговування автотранспорту має змогу змінити дані облікового запису, переглянути свої завдання, а також має можливість змінити статус роботи, що ним виконується та за необхідності додати чи видалити послуги, що необхідно надати клієнту.

Адміністратору доступний функціонал фахівця з ремонту, а також можливість додавати нових співробітників, редагувати та видаляти інформацію про існуючих працівників та послуги підприємства. Адміністратор може додавати, змінювати та видаляти облікові записи клієнтів, автомобілі та ремонтні роботи. Також адміністратор має можливість переглядати статистику з поточної діяльності підприємства та працівників.

Функціональна модель розроблюваної програмної системи представлена у вигляді діаграми варіантів використання. Модель варіантів використання наведено на рисунку 3.1.

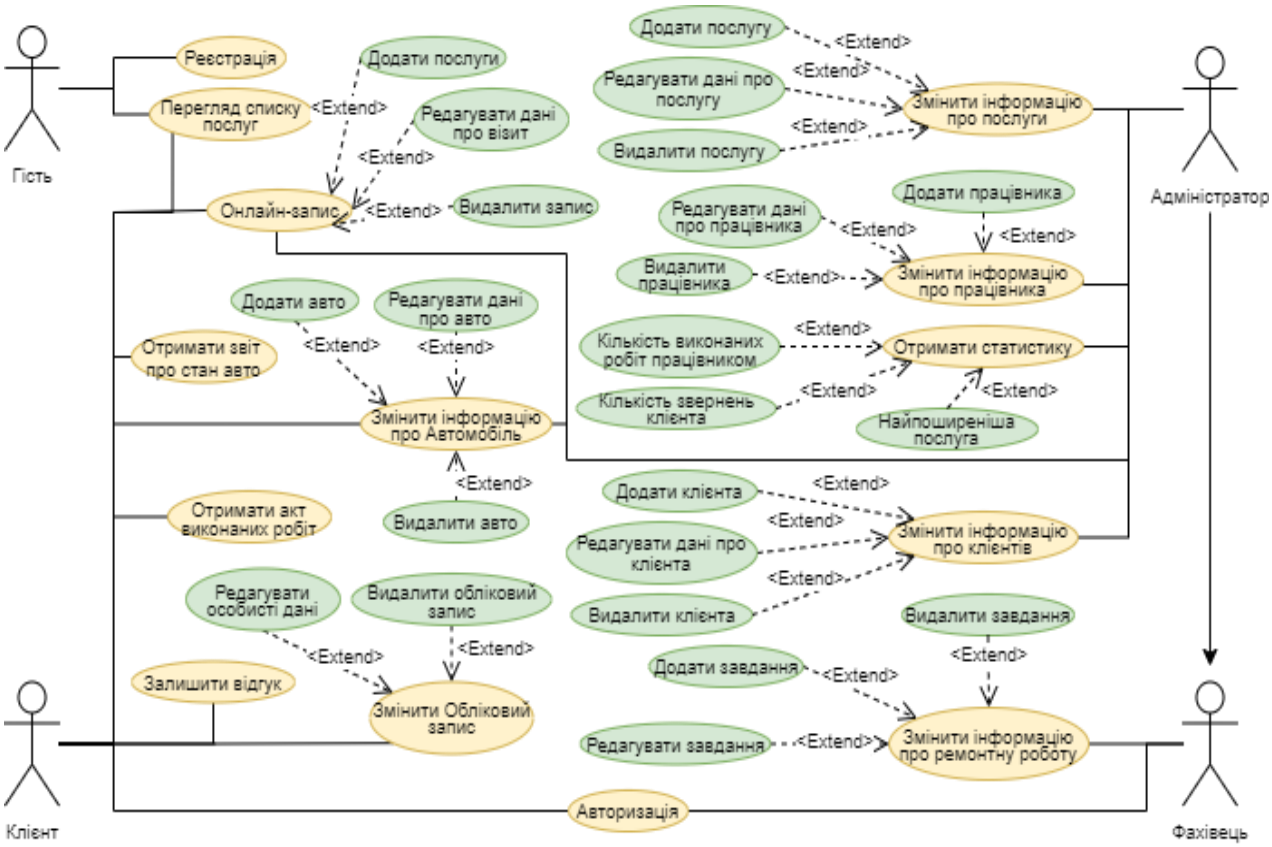


Рисунок 3.1 – Загальна USE-CASE діаграма (рисунок виконаний самостійно)

Під час моделювання поведінки програмної системи виникає необхідність деталізувати особливості алгоритмічної та логічної реалізації операції. Для цього використовуються блок-схеми, які акцентують увагу на послідовності виконання певних дій для досягнення визначеного результату [10].

З метою моделювання процесу виконання операцій використано діаграму активності або діяльності, яка зображується у вигляді графу вершинами якого є стани дії чи видів діяльності, а дуги – переходи від одного стану до іншого. Діаграма активності має початковий та кінцевий стани [10]. Діаграму активності зображено на рисунку 3.2.

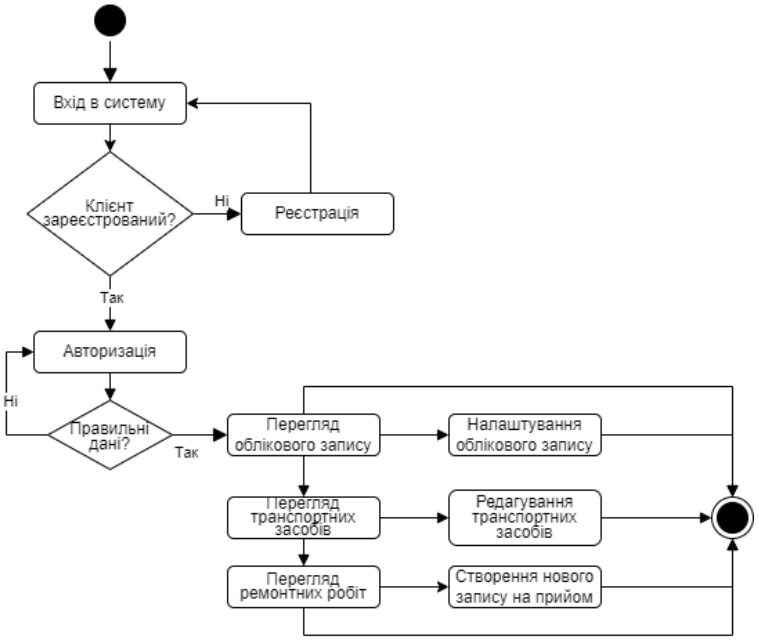


Рисунок 3.2 – Діаграма активності програмної системи (рисунок виконаний самостійно)

Отже, відкриття веб-додатку це початкова точка. У випадку якщо клієнт має обліковий запис, він має увійти в систему, якщо не має – то зареєструватися. Після авторизації в системі, клієнт потрапляє на сторінку облікового запису, звідки він може перейти до налаштувань облікового запису, де можна редагувати дані облікового запису чи видалити його, або до сторінки з інформацією про свої транспортні засоби. На сторінці транспортних засобів можна переглянути інформацію про транспортні засоби, також доступні можливості додавання, редагування, видалення даних про свої транспортні засоби, також можна перейти

до перегляду ремонтних робіт, де можна створити новий запис на прийом до фахівця або сформувати звіт виконаних робіт. З кожної сторінки користувач може вийти з системи.

З метою опису програмної системи було використано діаграму станів, яка показує як об'єкт переходить з одного стану в інший. Діаграму станів використовують при моделюванні життєвого циклу об’єкта. Вона відображає динамічні аспекти системи, визначає всі можливі стани, в яких може перебувати об'єкт, а також його поведінку, процес зміни станів об'єкта через вплив зовнішніх подій [10].

Програмна система передбачає можливість створення онлайн запису на прийом до фахівця, що передбачає створення нового об’єкту – запису на прийом. Діаграма станів зображена на рисунку 3.3.

Рисунок 3.3 – Діаграма станів програмної системи (рисунок виконаний самостійно)

За допомогою програмної системи користувач відкриває сторінку створення нового запису на прийом до фахівця знаходить та обирає потрібні послуги, вводить дату візиту, програмна система здійснює перевірку доступності обраної дати якщо дата доступна для запису система створює новий запис та призначає виконавця послуги та відображує результат користувачу та закриває сторінку створення запису та перенаправляє на головну сторінку облікового запису. У випадку неможливості створення запису система повідомляє про це користувача та

перенаправляє на головну сторінку облікового запису, надаючи можливість створити новий запис з іншими параметрами.

3.2 Проєктування архітектури ПЗ

Архітектура програмного забезпечення – це структура програми або обчислювальної системи, яка включає програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними.

В ході проєктування програмної системи було обрано клієнт-серверну архітектуру, що полягає у поділі на клієнтську та серверну частини. Для взаємодії з програмною системою користувач використовує робочу станцію – персональний комп’ютер або мобільний пристрій та інтернет-браузер. Основна частина веб-додатку знаходиться на сервері, а візуальна частина завантажується у браузері користувача. Клієнт формує запит на надсилає його на сервер за допомогою протоколу HTTP, сервер опрацьовує отриманий запит та повертає результат до клієнта. Сервер може опрацьовувати велику кількість запитів від багатьох клієнтів одночасно.

HTTP – протокол передачі даних, що використовується в комп'ютерних мережах та на основі якого працює мережа Інтернет. Це абревіатура від Hyper Text Transfer Protocol, протокол передачі гіпертекстових документів, що належить до протоколів моделі OSI 7-го прикладного рівня [11].

Застосування клієнт-серверної архітектури архітектура дозволяє розділити зони відповідальності між двома підсистемами і зробити незалежними. Для роботи з даними на серверній частині клієнтська частина програми повинна адресу URI запитуваного ресурсу. Кожен запит (HTTP-request) від клієнта до сервера повинен містити всю інформацію, необхідну для опрацювання запиту. Для створення запиту на сервер використовуються наступні HTTP-методи:

- GET – отримати ресурс із серверу;
- POST – створити ресурс на сервері;
- PUT – оновити ресурс на сервері;
- DELETE – видалити ресурс із серверу.

Серверна частина програми опрацьовує та надсилає відповідь HTTP-response на запит клієнта.

З метою відображення архітектури програмної системи було використано діаграму розгортання. Діаграма розгортання відображає архітектуру виконання системи, включаючи вузли, такі як апаратне або програмне середовище виконання, і проміжне програмне забезпечення, що з'єднує їх. Діаграми розгортання зазвичай використовуються для візуалізації фізичного обладнання та програмного забезпечення системи [10]. Діаграма розгортання наведено на рисунку 3.4.

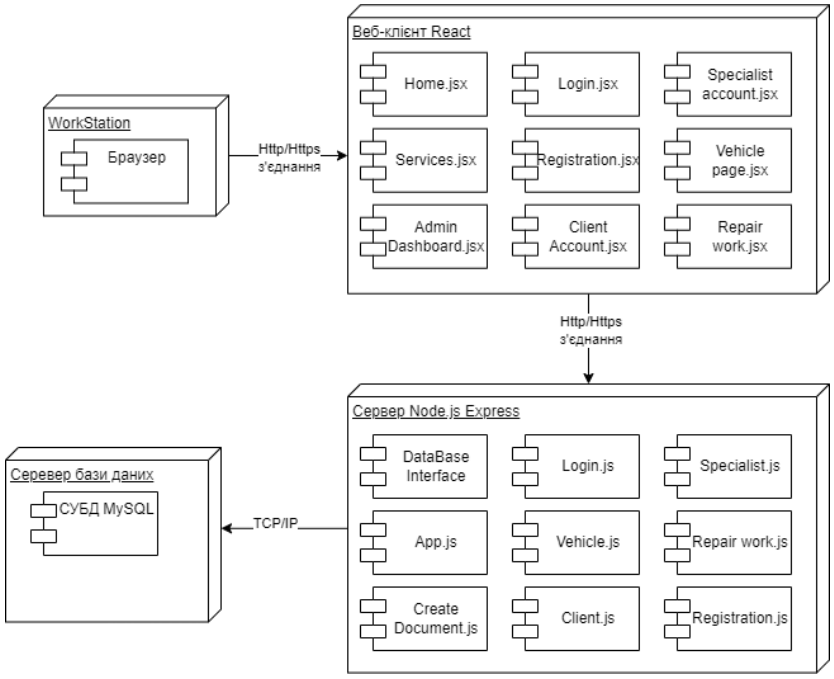


Рисунок 3.4 – Діаграма розгортання (рисунок виконаний самостійно)

Під час розробки програмної системи використано поняття «трирівнева архітектура», що є уособленням підходу «тонкий клієнт – товстий сервер». При використанні зазначеної архітектурі «тонкий» клієнт не перевантажений функціями обробки даних та виконує в основному основну роль в системі з відображення інформації, що надходить з сервера. Таких підхід зменшує обсяг даних, переданих між клієнтом і сервером та вимоги до характеристик клієнта.

Отже, програмна система складається з наступних елементів:

- клієнтом є веб-браузер встановлений на робочій станції, формує запити та надсилає їх до серверної частини системи;

- серверна частина здійснює обробку запиту та звертається до бази даних;
- база даних виконує операції з даними, що зберігаються в програмній системі.

Трирівнева архітектура клієнт-сервер дозволяє більш точно визначити ролі користувачів, так як вони отримують права доступу не до самої бази даних, а до певних функцій сервера додатків. Це підвищує захищеність системи не лише від навмисного пошкодження, а й від помилкових дій користувачів.

3.3 Проєктування структури зберігання даних

Після проведення аналізу вимог до програмної системи для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту та дослідження сутностей, що використані в моделі програмної системи було проведено семантичне моделювання, що являє собою моделювання структури даних, спираючись на зміст цих даних. В якості інструменту семантичного моделювання використовуються різні варіанти діаграм сутність-зв'язок (ER - Entity-Relationship). Так було створено ER-діаграму, яка відображує сутності предметної області та відношення та відношення між ними. ER-діаграму програмної системи наведено на рисунку 3.5.

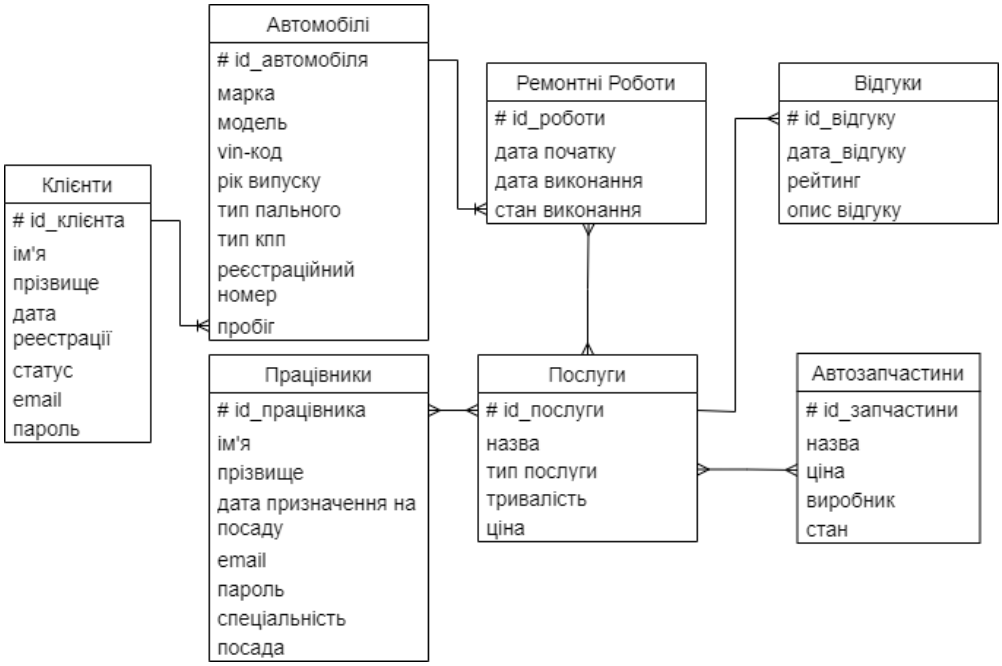


Рисунок 3.5 – ER-діаграма за нотацією Баркера (рисунок виконаний самостійно)

Для зберігання даних програмної системи було обрано реляційну модель бази даних, що дозволить зберігати дані у вигляді структурованих взаємопов’язаних таблиць. На основі ER-діаграми було розроблено логічну модель бази даних.

Схема бази даних програмної системи містить всього 9 таблиць, що використовують зв’язки один до багатьох та багато до багатьох. Таблиця «Клієнти» містить дані про клієнта такі як: id_клієнта унікальний ідентифікатор – первинний ключ, ім’я, прізвище, дата реєстрації, статус клієнта, електронна пошта та пароль, що використовуються для авторизації в системі.

Таблиця «Автомобілі» містить інформацію про транспортні засоби клієнта та пов’язана з таблицею «Клієнти» за допомогою зовнішнього ключа, (id_клієнта) використано зв’язок один до багатьох, тобто, один клієнт може мати декілька транспортних засобів. Атрибути сутності «Автомобіль»: id_автомобіля унікальний ідентифікатор, марка, модель, vin-код, рік випуску, тип пального, тип трансмісії, реєстраційний номер, пробіг зовнішнього ключа, (id_клієнта).

Таблиця «Працівники» містить дані про працівників підприємства такі як: id_працівника унікальний ідентифікатор – первинний ключ, ім’я, прізвище, дата призначення на посаду, спеціальність, посада, електронна пошта та пароль, що використовуються для авторизації в системі.

Таблиця «Автозапчастини» містить дані про автозапчастини, які можуть бути замінені під час надання послуг з ремонту та технічного обслуговування. Атрибути: id_запчастини, назва, ціна, виробник, стан.

Таблиця «Послуги» містить дані про послуги, які надаються підприємством з ремонту та технічного обслуговування автотранспорту .Атрибути: id_послуги, назва, ціна, тип послуги, тривалість. Так як один фахівець може виконати декілька послуг, а одна послуга може бути надана декількома фахівцями, застосовано зв’язок багато до багатьох тому було створено таблицю «Послуги для робіт» що поєднує таблиці «Послуги» та «Працівники». Так як під час надання однієї послуги може бути замінено багато запчастин, а одна запчастина може бути замінена під час надання декількох послуг, то використовується зв’язок багато до багатьох тому

було створено таблицю «Запчастини для послуг» що поєднує таблиці «Послуги» та «Автозапчастини».

Таблиця «Відгуки» містить дані про якість надання послуг та оцінку клієнтів. Атрибути: id_відгуку, дата відгуку, рейтинг, опис, зовнішній ключ для зв’язку з таблицею «Послуги». Для однієї послуги може бути надано декілька відгуків.

Таблиця «Ремонтні роботи» містить про проведені ремонтні роботи. Атрибути: id_роботи, дата початку, дата завершення стан виконання, id_автомобіля зовнішній ключ для зв’язку з таблицею «Автомобілі», id_послуги для робіт зовнішній ключ для зв’язку з таблицею «Послуги для робіт». Під час виконання однієї роботи може бути надано декілька послуг, а одна послуга може бути надана під час виконання декількох робіт, застосовано зв’язок багато до багатьох тому було створено таблицю «Послуги для робіт» що поєднує таблиці «Послуги» та «Ремонтні роботи». Для одного транспортного засобу може бути проведено декілька робіт, а одна робота може бути виконана лише для одного транспортного засобу, тому використовується зв’язок один до багатьох. Схему бази даних програмної системи наведено на рисунку 3.6.

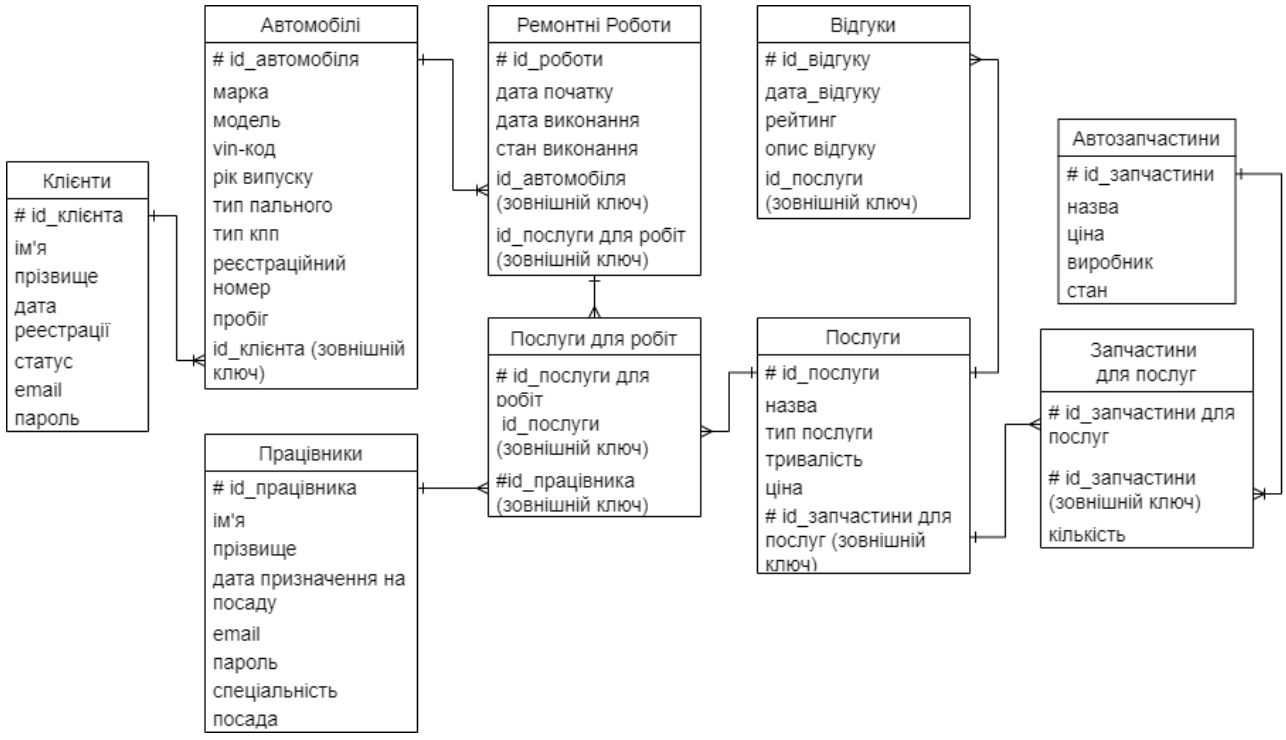


Рисунок 3.6 – Схема бази даних програмної системи (рисунок виконаний самостійно)

3.4 Приклади найцікавіших алгоритмів та методів

Програмна система для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту дозволяє автоматизувати задачу з розподілу завдань з ремонту та технічного обслуговування між працівниками – фахівцями з технічного обслуговування та ремонту. Задача автоматизації полягає у автоматичному підборі виконавця ремонтного завдання з урахуванням вартості ремонтної роботи та зайнятості майстра. Так, наприклад у випадку вартості роботи більше 10000 грн. її може виконувати працівник 5 розряду, а якщо вартість роботи менше 2000 грн, то її може виконувати фахівець початківець першого розряду.

Користувач у вікні «Ремонтні роботи» обирає дату проведення ремонтної роботи, обирає зі списку назву ремонтної роботи. Після натискання користувачем кнопки «Створити запис» програмна система на знаходить фахівця який може виконати обране завдання з урахуванням вартості завдання та досвідом майстра також перевіряється доступність майстра, якщо в обраний час він немає іншого завдання. Якщо фахівця знайдено здійснюється додавання нового запису та сповіщення користувача про результат. У випадку відсутності фахівців у обрану дату користувач отримає сповіщення з пропозицією обрати іншу дату. Діаграма активності зображена на рисунку 3.7.

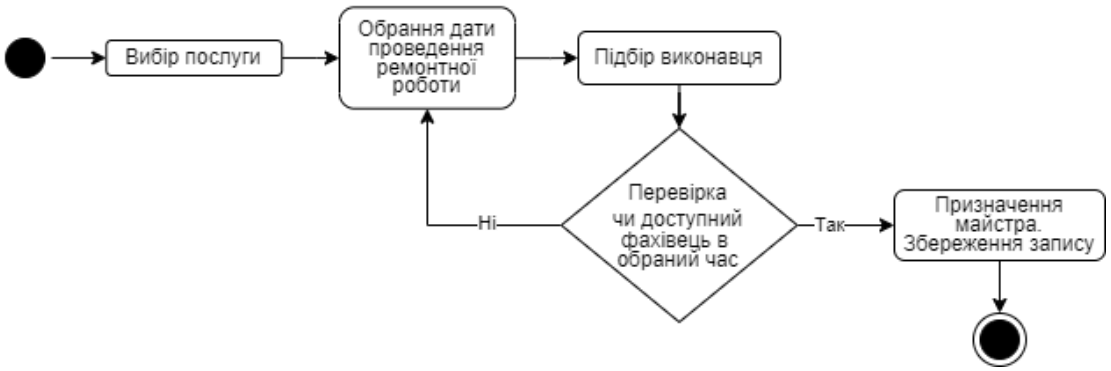


Рисунок 3.7 – Діаграма активності програмної системи призначення виконавця (рисунок виконаний самостійно)

Отже, за допомогою програмної системи користувач знаходить та обирає потрібні послуги, вводить дату візиту, програмна система здійснює перевірку

доступності обраної дати якщо дата доступна для запису система створює новий запис та призначає виконавця послуги та відображує результат користувачу, у випадку неможливості запису в обраний день система пропонує обрати іншу дату.

3.5 Створення UI / UX дизайну системи

Користувач взаємодіє з програмною системою за допомогою інтерфейсу користувача. З метою забезпечення ефективної взаємодії інтерфейс користувача має бути інтуїтивно зрозумілим, а кольорова схема, що використовується в оформленні графічного інтерфейсу не повинна викликати втому від роботи з системою. В процесі розробки графічного дизайну інтерфейсу користувача застосовуються прототипи сторінок веб-застосунку. Створення прототипів – детальних макетів сторінок дає можливість візуалізувати дизайн веб-додатку та оцінити наскільки зручним він буде для користувача.

Отже, було розроблено прототипи для ключових сторінок програмної системи. Для того щоб скористуватися функціоналом програмної системи користувачу необхідно авторизуватися. Для цього розроблено сторінку авторизації зображення якої наведено на рисунку 3.8.

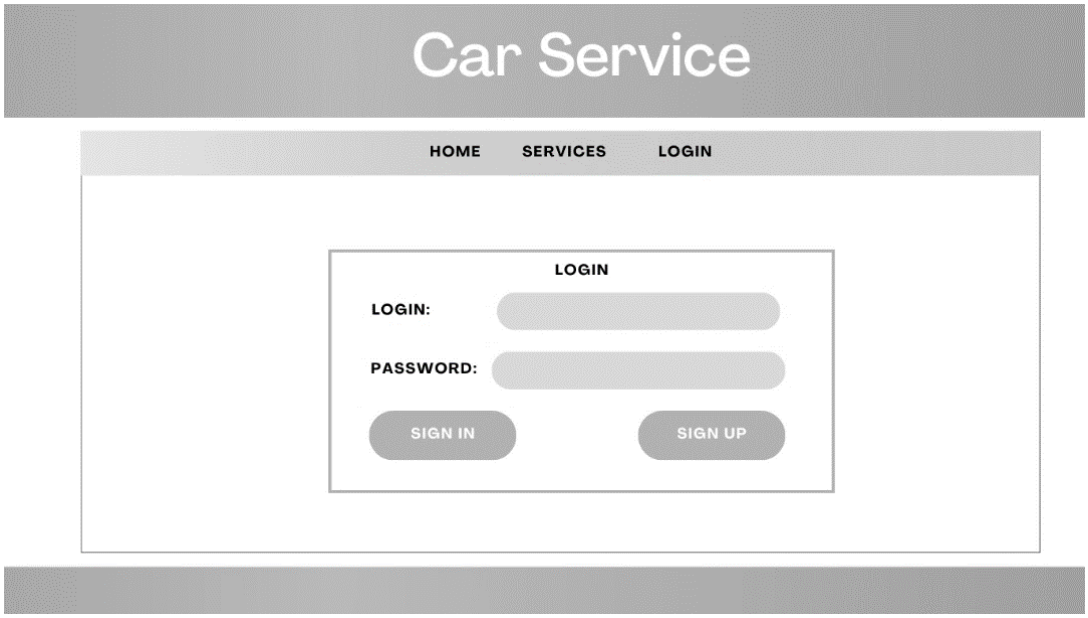


Рисунок 3.8 – Сторінка входу в систему (рисунок виконаний самостійно)

Для адміністратора програмної системи передбачено сторінку, де можна переглядати відомості про працівників, послуги, ремонтних робіт, клієнтів та за необхідності здійснювати редагування та видалення даних. Зображення сторінки адміністратора наведено на рисунку 3.9.

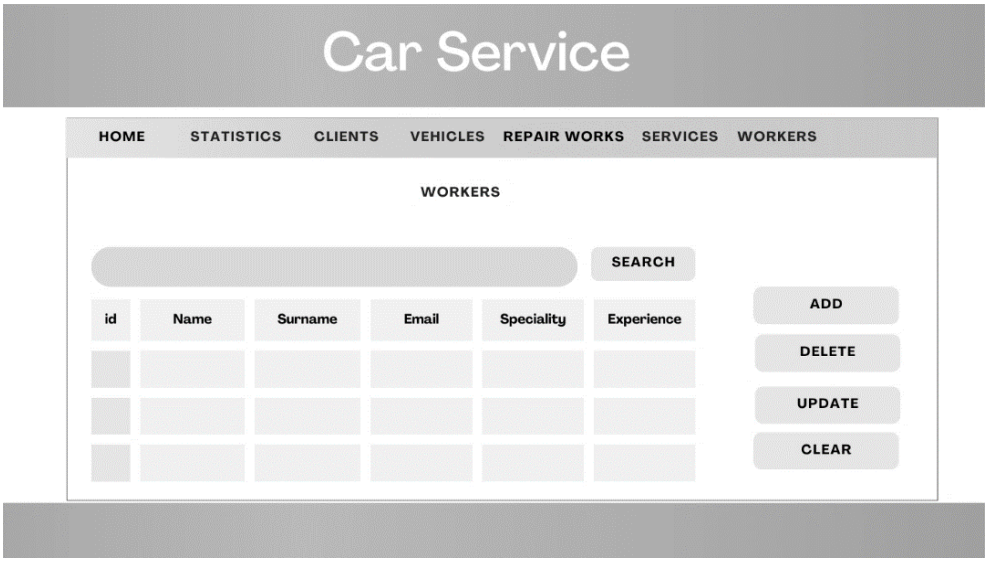


Рисунок 3.9 – Сторінка адміністратора програмної системи (рисунок виконаний самостійно)

Фахівець з ремонту та технічного обслуговування має особисту сторінку на якій відображають завдання з ремонту та технічного обслуговування, а також доступні дії з додавання видалення та редагування записів про ремонтні роботи. Зображення сторінки адміністратора наведено на рисунку 3.10.

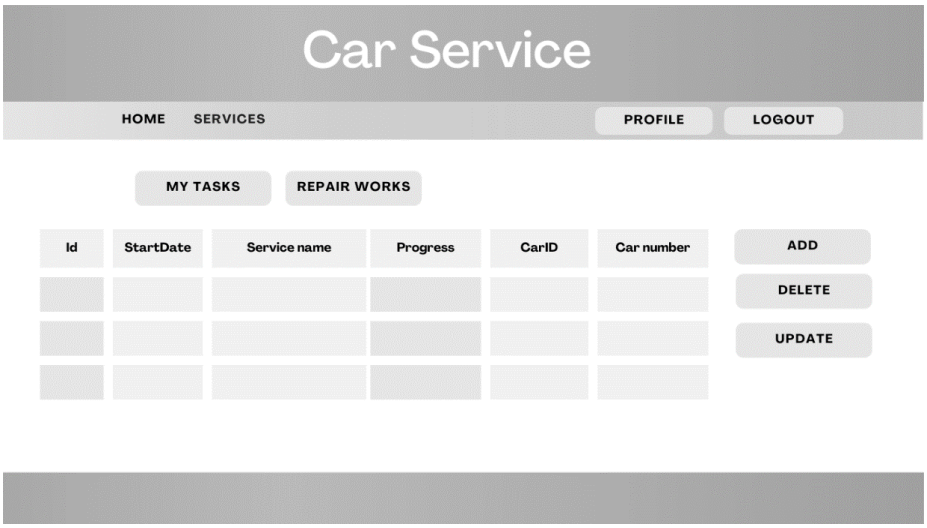


Рисунок 3.10 – Особиста сторінка фахівця з технічного обслуговування (рисунок виконаний самостійно)

Програмна система передбачає можливість для клієнта здійснювати онлайн-запис на прийом до фахівця з ремонту. Для цього розроблено сторінку особистий кабінет клієнта, де можна створити новий запис та за необхідності здійснювати редагування та видалення даних про запис на прийом до фахівця з ремонту та технічного обслуговування автотранспорту. Зображення сторінки для онлайн-запису наведено на рисунку 3.11.

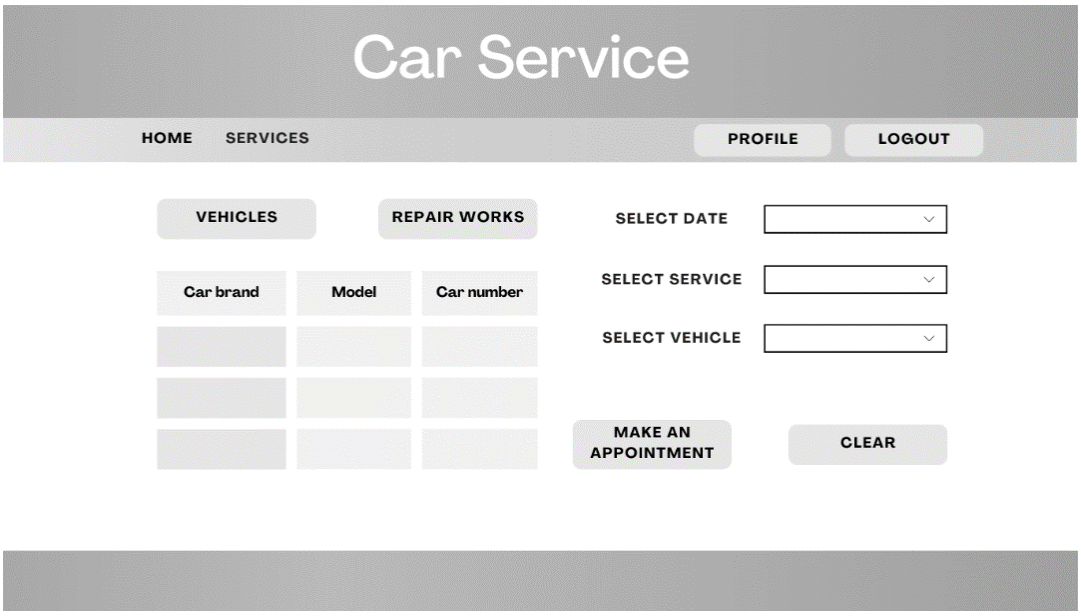


Рисунок 3.11 – Сторінка особистого кабінету клієнта (рисунок виконаний самостійно)

Отже, створені прототипи інтерфейсу користувача відображають основні сторінки програмної системи та дозволяють оцінити функціональність та зручність використання програмної системи.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Опис реалізації серверної частини

З метою створення серверної частини програмної системи було обрано Node.js та фреймворк Express.js.

Node.js – це програмне середовище, яке дозволяє запускати програми, написані мовою Javascript, поза браузером. Історично програми, написані на Javascript, на відміну від інших мов програмування, можна було запустити лише у браузерах, які мали спеціальний вбудований движок виконання коду цієї мови. При розробці Node.js за основу було взято двигун виконання JavaScript під назвою V8, який був створений компанією Google і використовувався в браузері Google Chrome. Node.js має відкритий код, тому працювати з ним можна абсолютно безкоштовно. Його і сьогодні продовжує розвивати та покращувати глобальна спільнота розробників. Node.js здобув широку популярність серед розробників веб-застосунків та API завдяки своїй ефективності та масштабованості та легкості у встановленні та використанні [12].

До особливостей Node.js відносяться:

- використання асинхронної архітектури, котра надає можливість здійснювати оброблення великої кількості запитів одночасно без блокування виконання інших операцій, завдяки використанню non-blocking I/O операцій введення та виведення та callback функцій;
- використання одного потоку для обробки запитів, завдяки механізму подій (Event Loop) Node.js здатний обробляти багато запитів одночасно;
- наявність менеджеру пакетів NPM, котрий надає можливість простого та швидкого встановлення та управління залежностями проекту. Багато сторонніх бібліотек та модулів доступні через NPM, що робить його важливою частиною екосистеми Node.js;
- кросплатформність, Node.js може бути запущено на різних операційних системах (Windows, Linux, macOS) [13].

Отже, Node.js являє собою парадигму «JavaScript everywhere», яка об'єднує розробку веб-програм за єдиною мовою програмування, а не різними мовами для

створення скриптів на стороні сервера та клієнта.

Express.js. – це фреймворк для Node.js, що використовується для розробки веб-застосунків з серверною стороною та API. Експрес забезпечує простий інтерфейс для створення веб-застосунків. Особливостями фреймворку Express є:

- маршрутизація, наявність зручного механізму для визначення маршрутів HTTP-запитів;
- підтримка шаблонів, можливість інтеграції з різними шаблонізаторами (Jade, Handlebars і EJS);
- концепція проміжного ПЗ (Middleware), що дозволяє обробляти запити до того, як вони досягають маршрутів, що дозволяє виконувати різні операції, наприклад, аутентифікація;
- фреймворк забезпечує просту інтеграцію з різними базами даних (MySQL, MongoDB та PostgreSQL) [14].

Для зберігання даних програмної системи було використано реляційну базу даних СКБД MySql. Для роботи веб-застосунку з базою даних використано Sequelize.

Sequelize – це (ORM) бібліотека об’єктно-реляційна проекція для Node.js, котра забезпечує рівень абстракції над реляційними базами даних, полегшуючи взаємодію з базами даних за допомогою об’єктів JavaScript замість запитів SQL. Sequelize підтримує різні системи баз даних, такі як MySQL, PostgreSQL, SQLite.

Sequelize дозволяє вам визначати моделі даних, виконувати запити до бази даних і автоматично трансформувати дані між форматом JavaScript і форматом SQL, що використовується в базі даних [15].

ORM Object-relational mapping (Об’єктно-реляційне відображення) – це концепція, яка спрощує взаємодію між об’єктно-орієнтованим програмуванням і реляційними базами даних. Основні переваги ORM:

- спрощення взаємодії з базою даних, не потрібно писати SQL-запити вручну так як робота з даними відбувається за допомогою об’єктів та методів;
- наявність вбудованих методів для створення, читання, редагування та видалення записів в базі даних;

- забезпечення абстракції бази даних, що дозволяє розробникам змінювати тип бази даних без значних змін в програмному коді;
- дозволяє визначати відносини між об'єктами (наприклад, один до багатьох, багато до багатьох), що дозволяє легко працювати зі складними структурами даних.

Отже, для підключення до бази даних за допомогою Sequelize необхідно зазначити ім’я бази даних, користувача, пароль доступу, хост – адресу, де розташована бази даних в даному випадку вона розташована на комп’ютері тож використано localhost та діалект бази даних в даному випадку це mySql:

```
const database = new Sequelize(process.env.DB_NAME,
process.env.DB_USER, process.env.DB_PASSWORD, {
    host: process.env.DB_HOST,
    dialect: process.env.DB_DIALECT,
});
export default database;
```

Для обробки маршрутів в застосунку Node.js використано app.use(), що підключає middleware для обробки маршрутів (clientRoute, employeeRoute тощо) до відповідного маршруту (/api/clients, /api/employees і т.д.). Ця структура організовує API-точки доступу за визначеними маршрутами. Middleware – проміжний обробник, який дозволяє обробляти запити до сервера перед тим, як вони досягнуть обробника запитів (route handler) або відправлять відповіді клієнту після обробки.

```
app.use('/api/auth', authRoute)
app.use('/api/clients', clientRoute)
app.use('/api/employees', employeeRoute)
app.use('/api/services', serviceRoute)
app.use('/api/carparts', carPartRoute)
app.use('/api/works', maintenanceWorkRoute)
app.use('/api/vehicles', vehicleRoute)
```

Для того щоб взаємодіяти з сутностями бази даних створювати, редагувати, видаляти, обирати за параметрами було створено окремі функції? доступ до яких здійснюється за визначеними маршрутами.

Створення нового запису відбувається за допомогою методу POST, редагування запису здійснюється за допомогою PUT, видалення запису за

допомогою DELETE, для отримання запису, наприклад усіх записів що задовольняють якусь умову використовується GET.

Нижче наведено приклад маршрутів запитів для створення, редагування та видалення, отримання всіх та пошуку за первинним ключем транспортних засобів клієнта.

```
router.get('/vehicles/edit/:id',    authenticate,    restrict(['admin',
'specialist']), getVehicleById);
router.get('/vehicles/edit/',     authenticate,     restrict(['admin',
'specialist']), getAllVehicles);
router.post('/vehicles/edit',     authenticate,     restrict(['admin',
'client']), createVehicle);
router.put('/vehicles/edit/:id',    authenticate,    restrict(['admin',
'client']), updateVehicle);
router.delete('/vehicles/edit/:id',  authenticate,  restrict(['admin',
'client']), deleteVehicle);
```

З метою забезпечити захист від несанкціонованого доступу до виконання тієї чи іншої дії (видалити, створити запис тощо) було передбачено авторизацію користувача в програмній системі. Для цього було використано бібліотеку jsonwebtoken та створено функцію відповідну функцію яка перевірятиме наявність токену та він починається з рядка «Bearer». Якщо умова не виконується, то функція повертає відповідь з HTTP статусом 401, що означає що користувач неавторизований та не може виконати подальші дії.

Якщо токен існує та має правильний формат токен перевіряється за допомогою jwt.verify з використанням секретного ключа, який зберігається у змінній середовища process.env.JWT_SECRET_KEY. Якщо перевірка успішна, дані ідентифікатора користувача (decoded.id) і ролі користувача (decoded.role) додаються до запиту (об’єкт req) для подальшого використання. Нижче наведено код функції authenticate:

```
export const authenticate = async (req, res, next) => {
    const authToken = req.headers.authorization
    if (!authToken || !authToken.startsWith('Bearer ')) {
        return res.status(401).json({ success: false, message: "No
token, authorisation denied!" })
    }
    try {
```

```
        const token = authToken.split(" ")[1];
        const decoded = jwt.verify(token, process.env.JWT_SECRET_KEY)
        req.userId = decoded.id
        req.role = decoded.role
        next()
    } catch (error) {
        if (error.name == 'TokenExpiredError') {
            return res.status(401).json({ message: 'Token is expired!'
})
        }
       return res.status(401).json({ success: false, message: 'Invalid
token!' })
    }};
```

Також для того щоб перевірити чи має авторизований користувач права на здійснення певної дії передбачено розподіл користувачів за ролями:

- клієнт;

- фахівець;

-адміністратор.

Відповідні ролі присвоюються під час реєстрації в програмній системі. Перевірка здійснюється прав користувача здійснюється за допомогою наступного коду:

```
export const restrict = roles => async (req, res, next) => {
    const userRole = req.role
    if (!roles.includes(userRole)) {
         return res.status(401).json({ success: false, message: 'Not
authorized!', data: userRole });    }
    next();};
```

Реєстрація нового працівника здійснюється адміністратором програмної системи, клієнт підприємства з ремонту та технічного обслуговування має можливість зареєструватися самостійно. Функція для реєстрації нового клієнта:

```
export const register = async (req, res) => {
const { name, surname, email, address, phone, status, password } =
req.body;
  try {
    let client = null;
    client = await Client.findOne({
      attributes: ['id', 'name', 'surname', 'email'],
      where: { email: req.body.email } });
if (client) {
return res.status(400).json({ message: "User already exists" });}
```

```
const salt = await bcrypt.genSalt(10);
const hashPassword = await bcrypt.hash(password, salt);
client = await Client.create({
      name: name, surname: surname, email: email, address: address,
      phone: phone, status: status, password: hashPassword,});
res.status(200).json({ success: true, message: "User was successfully
created!" });
} catch (error) {
    res.status(500).json({ success: false, message: "Server Error!
 User was not created! Try again!" });
}};
```

За допомогою зазначеної функції спочатку здійснюється перевірка чи існує вже клієнт з зазначеною електронною поштою, у випадку якщо такий користувач існує функція повертає відповідне повідомлення, в протилежному випадку здійснюється реєстрація нового клієнта в системі. Збереження паролю для входу в програмну систему здійснюється у захешованому вигляді. Для цього використано бібліотеку bcrypt.

Для входу в програмну систему користувачу необхідно мати пароль та електронну пошту. Користувач вводить ці дані у відповідній формі, потім надсилає запит на сервер, де перевіряється чи існує користувач з такою електронною поштою, якщо так, то порівнюються надісланий пароль та збережений у захешованому вигляді в базі даних, при відповідності паролів користувач входить в систему перевірка пар паролів здійснюється за допомогою бібліотеки bcrypt, вбудований метод порівняння (compare).

Для створення JWT (JSON Web Token) на основі інформації про користувача, використовується функція generateToken():

```
const generateToken = user => {
return    jwt.sign({    id:    user.id,    role:    user.role    },
process.env.JWT_SECRET_KEY, {
  expiresIn: process.env.JWT_REFRESH_EXPIRATION,
  })}
```

Функція jwt.sign генерує JWT з інформацією про користувача його роль та ідентифікатор, підписує його за допомогою секретного ключа і встановлює термін дії токену. Отриманий токен використовується для автентифікації та авторизації користувача в програмній системі.

Для того щоб встановити відносини між таблицями в базі даних за допомогою Sequelize використовується наступний запис:

```
Client.hasMany(Vehicle, { as: 'vehicles', foreignKey: 'clientId'});
Vehicle.belongsTo(Client, { as: 'clients', foreignKey: 'clientId'});
```

Таким чином створено зв’язок один-до-багатьох, один клієнт може мати декілька автомобілів. Для встановлення відношення багато до багатьох використовується belongsToMany(). Нижче створено зв’язок багато-до-багатьох між таблицями «Ремонтні роботи» та «Послуги», адже під час виконання однієї ремонтної роботи може бути надано декілька послуг, а одна послуга може бути надана під час виконання декількох ремонтних робіт.

```
MaintenanceWork.belongsToMany(Service, { through:
ServiceTask,  foreignKey: 'serviceId' });
Service.belongsToMany(MaintenanceWork, { through:
ServiceTask,  foreignKey: 'workId' });
```

Отже, створена серверна частина програмної системи дозволяє взаємодіяти з базою даних, здійснювати авторизацію та реєстрацію в програмній системі додавати, редагувати та видаляти дані про клієнтів, працівників, транспортні засоби, послуги підприємства, здійснювати пошук та призначати фахівців для виконання робіт з технічного обслуговування та ремонту.

4.2 Опис реалізації клієнтської частини

Для створення клієнтської частини було використано бібліотеку React.

React – це JavaScript бібліотека, яка використовується для створення інтерфейсу користувача (UI). Була розроблена компанією Facebook основне призначення якої створення для побудови Single Page Applications одно сторінкових веб-застосунків [16].

До особливостей бібліотеки React відносяться:

- компонентна архітектура, веб-застосунок складається з малих незалежних один від одного компонентів, що забезпечує повторне використання коду та зменшує обсяг веб-застосунку;

- віртуальний Document Object Model (DOM) для оновлення та відображення змін у інтерфейсі користувача, шо забезпечує оптимізацію рендерингу сторінок;

- можливість створення одно сторінкових веб-застосунків (Single Page Applications), що дозволяє змінювати наповнення сторінки без необхідності її перезавантаження [16].

Для забезпечення можливості взаємодії застосунку React з додатком Node.js було використано бібліотеку для здійснення HTTP-запитів з JavaScript – Axios. Axios надає зручний інтерфейс для виконання запитів на сервер та обробки відповідей. Axios підтримує безліч методів запитів, включаючи GET, POST, PUT, DELETE та інші. Він також підтримує завантаження та відправлення файлів, встановлення заголовків запитів [17].

Нижче наведено приклад створення запиту для отримання усіх послуг підприємства за допомогою методу GET та встановлення заголовку з авторизацією:

```
const response = await axios.get(`${BASE_URL}/services/services/all`,
{
          headers: {'Authorization': `Bearer ${token}`,}
});
```

Для відправлення запиту створення нового запису в базі даних з додатку React використовується метод POST. Нижче наведено приклад додавання нового транспортного засобу:

```
response=    await   axios.post(`${BASE_URL}/vehicles/vehicles/edit`,
editedVehicle,{
            headers: {'Authorization': `Bearer ${token}`, }
} );
```

З метою забезпечити захист паролю під час відправлення його на сервер використовується бібліотека crypto-js. Після введення у відповідну форму пароль користувача шифрується за допомого функції:

```
const hashedPassword = CryptoJS.MD5(formData.password).toString();
```

Для того щоб не передавати на сервер не коректні дані було створено валідацію даних в React-застосунку. Прикладом валідації може бути перевірка що електронна пошта записана користувачем у правильному форматі:

```
const validateEmail = (email) => {
    const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-
Z]{2,}$/;
    return emailRegex.test(email);
  };
```

На головній сторінці програмної системи користувач бачить назву веб-додатку та кнопки вхід та розпочати роботу. У випадку якщо користувач вже авторизований, замість кнопки «Вхід» він побачить своє ім’я та прізвище та посилання на особистий кабінет, кнопка «Розпочати роботу» також відкриє сторінку особистого кабінету, якщо користувач не увійшов до програмної системи після натискання кнопки «Розпочати роботу» його буде направлено на сторінку реєстрації. Рисунок головної сторінки зображено на рисунку 4.1.

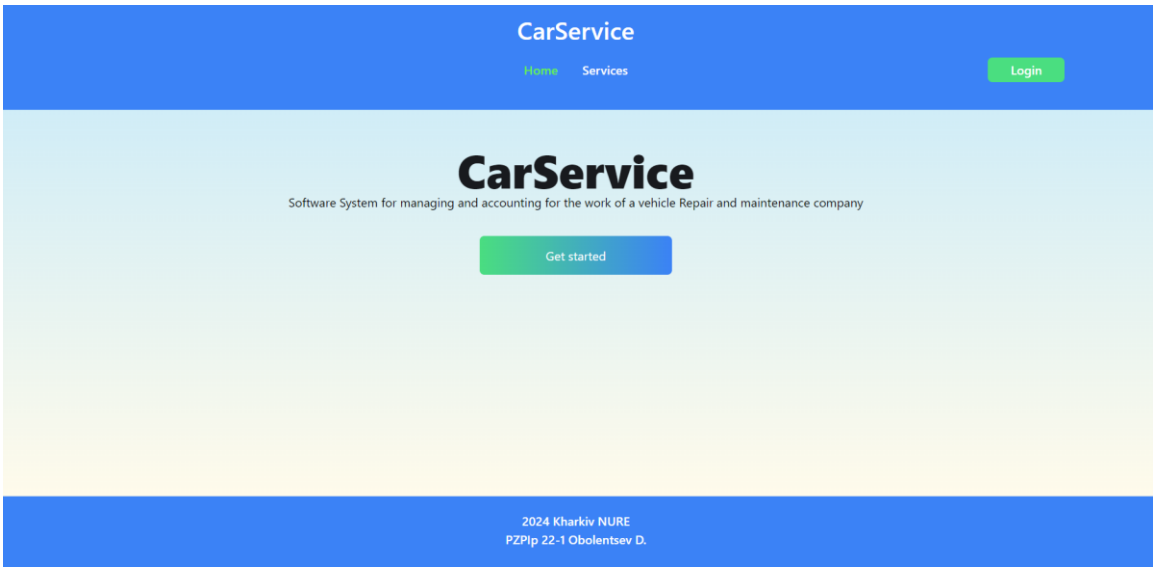


Рисунок 4.1 – Головна сторінка (рисунок виконаний самостійно)

На сторінці входу в систему користувач повинен заповнити форму ввести електронну пошту та пароль. Якщо заповнені дані правильно, такий користувач зареєстрований в системі, то відбудеться перехід на особисту сторінку відповідно до його ролі (клієнт, фахівець, адміністратор) На рисунку 4.2 зображено сторінку клієнта, де відображено особисті дані користувача, його транспортні засоби, та кнопки для виходу з системи, переходу на сторінки редагування профілю, де можна змінити особисті дані, пароль для входу та видалити обліковий запис, сторінка «Роботи з технічного обслуговування», де розташовані записи про всі візити та ремонтні роботи, що проводяться чи були проведені для транспортних засобів клієнта. Також можна перейти на сторінку «Звіти», де можна сформувати акт виконаних робіт та «Відгуки», де можна залишити свій відгук.

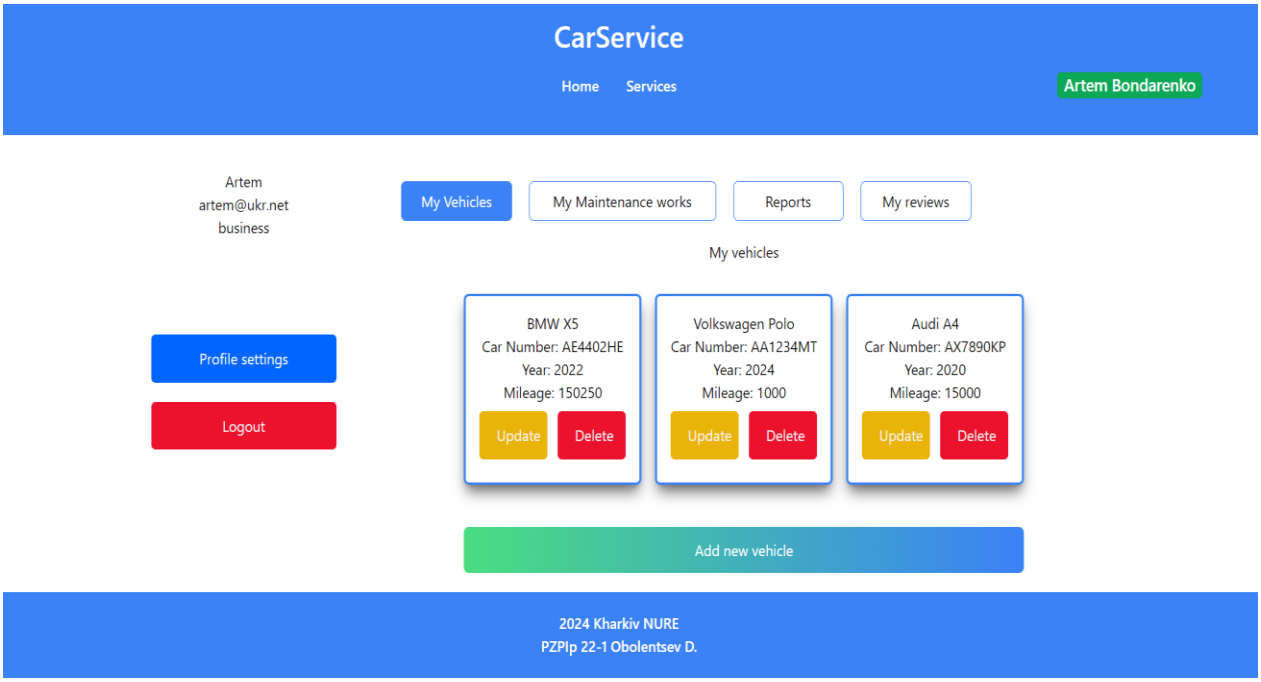


Рисунок 4.2 – Сторінка клієнта (рисунок виконаний самостійно)

На сторінці «Ремонтні роботи» можна здійснити пошук за датою виконання для цього необхідно заповнити поле пошуку та натиснути кнопку пошук, для того щоб здійснити фільтрацію записів про ремонтні роботи за транспортним засобом потрібно з випадного списку обрати необхідний транспортний засіб також доступне сортування за датою виконання.

На сторінці також можливо видалити, редагувати чи створити новий запис про ремонтну роботу натиснувши відповідні кнопки. На рисунку 4.3 зображено сторінку «Ремонтні роботи».

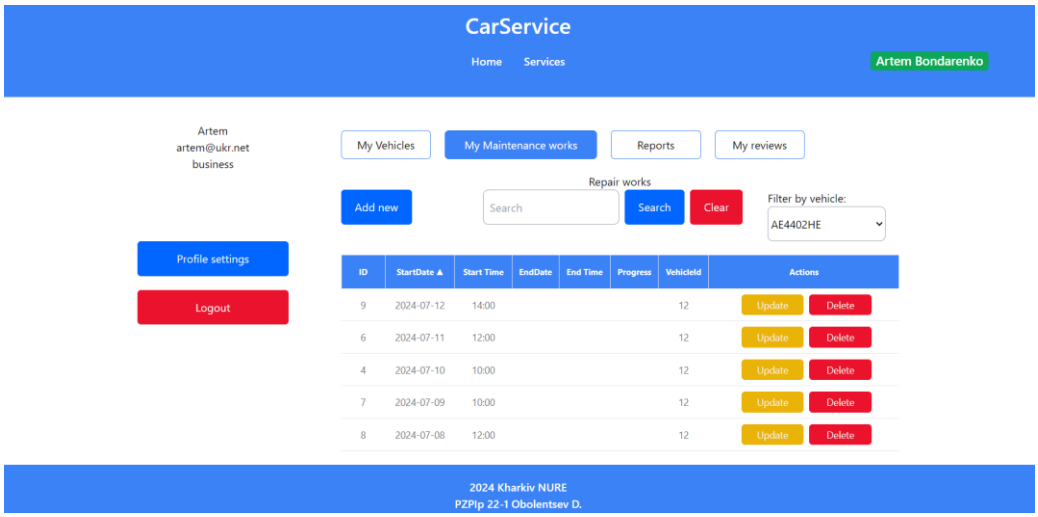


Рисунок 4.3 – Сторінка «Ремонтні роботи» (рисунок виконаний самостійно)

Після натискання створити новий запис відкривається модальне вікно з формою для створення нового запису. Якщо натиснути кнопку «Редагувати» відкриється це ж вікно завантаженими у форму записами, які необхідно редагувати. Після натискання додати створюється новий запис, також тут можна обрати зі списку послуги, які необхідно виконати, обрані послуги будуть відображені у таблиці. Вікно створення та редагування запису про ремонтні роботи зображено на рисунку 4.4

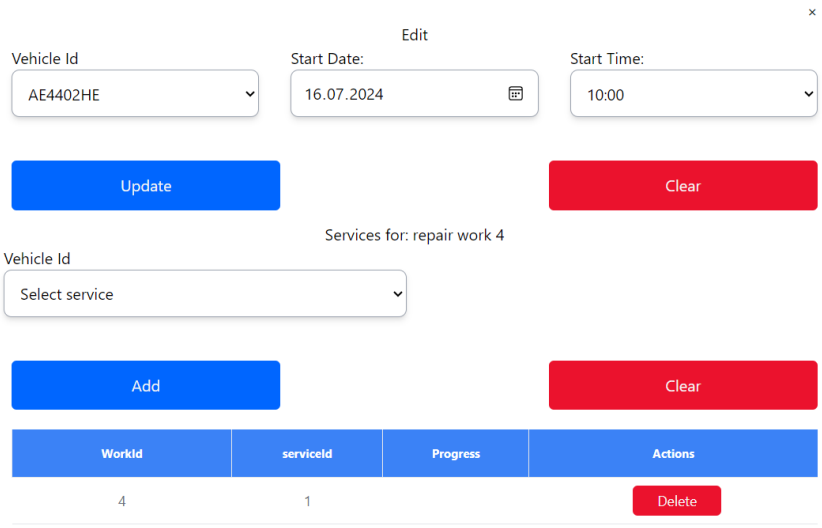


Рисунок 4.4 – Вікно додавання нового запису (рисунок виконаний самостійно)

Для адміністратора передбачено окрему сторінку котра дозволяє здійснювати роботу з базою даних, додавати редагувати, видаляти інформацію про клієнтів, працівників, послуги тощо, а також отримувати статистику з поточної діяльності підприємства. Фахівець з ремонту на своїй сторінці може відслідковувати завдання з ремонту та змінювати їх статус. На рисунку 4.5 зображена сторінка адміністратора програмної системи. На рисунку 4.6 зображена сторінка фахівця з ремонту.

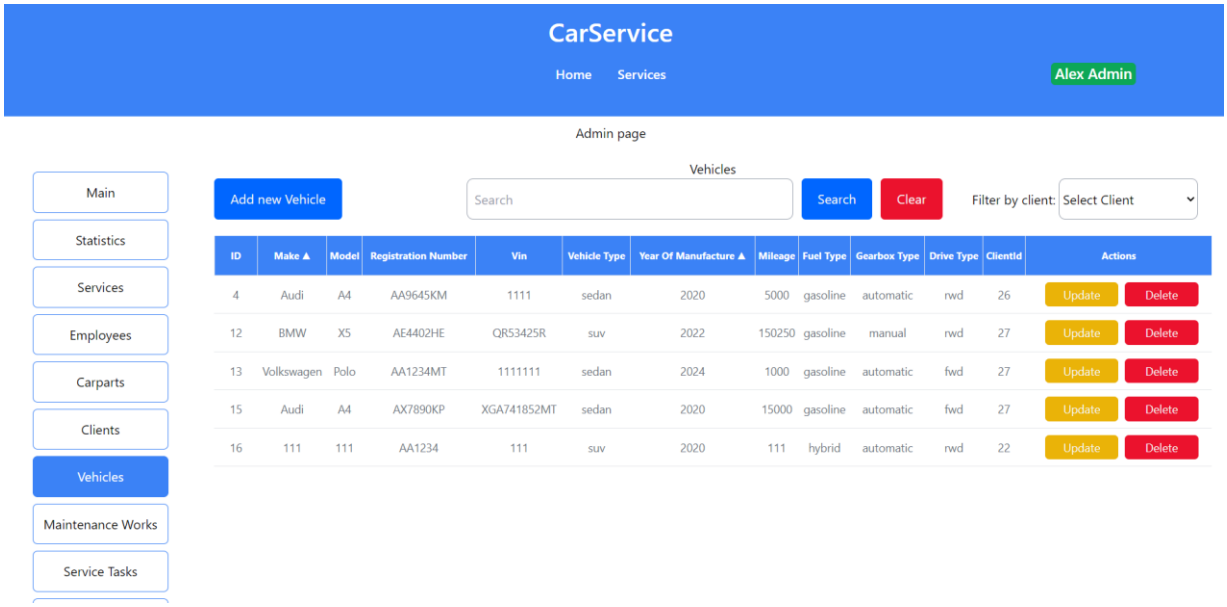


Рисунок 4.5 – Сторінка адміністратора системи (рисунок виконаний самостійно)

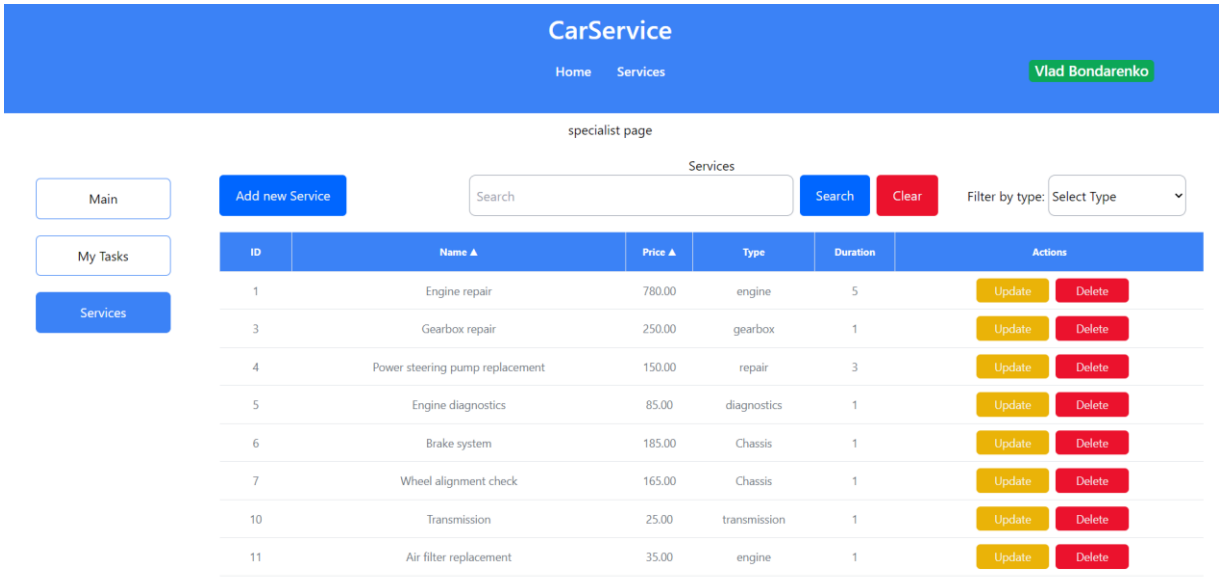


Рисунок 4.6 – Сторінка фахівця з ремонту (рисунок виконаний самостійно)

У випадку якщо користувач використовує для входу в програмну систему мобільний пристрій, то дизайн сторінки адаптується під розміри екрану. На рисунку 4.7 зображено сторінку адміністратора відкриту на мобільному пристрої.

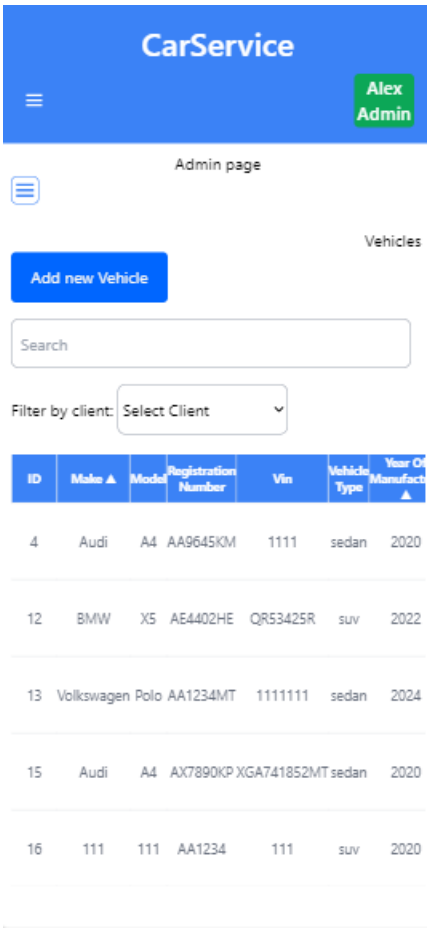


Рисунок 4.7 – Сторінка адміністратора відкрита на мобільному пристрої (рисунок виконаний самостійно)

Отже, створена клієнтська частина надає зручний інтерфейс користувача, що дозволяє ефективно взаємодіяти з серверною частиною та дозволяє клієнтам здійснювати запис на прийом до фахівця з ремонту, фахівцям відслідковувати поточні завдання та змінювати статус завдань, а адміністратору здійснювати облік клієнтів та транспортних засобів та здійснювати розподіл завдань з ремонту та технічного обслуговування між фахівцями з урахуванням їх досвіду та зайнятості.

5 ТЕСТУВАННННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

З метою виявлення ролі тестування в процесі розробки програмного забезпечення варто розглянути теоретичне визначення поняття «тестування програмного забезпечення». Так, в наукових джерелах поняття тестування програмного забезпечення визначається як: «вид діяльності в процесі розробки програмного забезпечення, пов'язаний з виконанням процедур, спрямованих на виявлення помилок в поточному визначенні розроблюваної програмної системи. Процес тестування стосується в першу чергу перевірки коректності програмної реалізації системи, відповідності реалізації вимогам, тобто тестування – це кероване виконання програми з метою виявлення невідповідностей її поведінки та вимог до програмного забезпечення» [18].

Отже, тестування програмного забезпечення відіграє важливу роль в життєвому циклі програмного забезпечення, дозволяє виявити дефекти, помилки та інші невідповідності до сформованим вимогам до програмної системи.

«При проведенні тестування завжди має місце «зовнішній вплив» на систему з боку тестувальника, який також називають «тестовим впливом». Тестування полягає у здійсненні тестувальником спрямованого впливу на систему, що тестується, який передбачає отримання деякої очікуваної реакції, яку називають «тестовою реакцією», поява якої має підтвердити відповідність системи конкретної окремо взятої специфікації. Отримання протилежної (стосовно очікуваної) або неспецифічної реакції говорить про невідповідність системи специфікації і необхідність повернення програмного продукту на доопрацювання з метою усунення дефекту» [18].

В залежності від цілей тестування, виділяють наступні види тестування програмного забезпечення:

- функціональне тестування;
- нефункціональне тестування;
- регресійне тестування;
- структурне тестування [18].

Для тестування програмної системи було обрано функціональне тестування, що полягає у перевірці коректності виконуваних системою функцій, тобто виявлення невідповідностей між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і вимог до програмної системи. Функціональні тести повинні охоплювати всі реалізовані функції з урахуванням найбільш ймовірних типів помилок. Перевагою функціонального тестування є імітація фактичного використання системи.

Для тестування було обрано наступні функції програмної системи:

- реєстрація нового клієнта;
- додавання нового транспортного засобу;
- оновлення інформації про транспортні засоби;
- видалення інформації про ремонтні роботи.

Було розроблено відповідні тест-кейси які наведено в таблицях 5.1 – 5.4.

Таблиця 5.1 – Тест-кейс 1 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	ТС 1		
Опис функції:	Створення нового облікового запису		
Власник тесту:	Оболенцев Денис		
Дата створення:	20.06.2024		
Мета тесту:	Перевірити коректність створення нового облікового запису клієнта		
Створення нового облікового запису			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити сторінку реєстрації	З’являється форма для реєстрації в системі	Пройдено
2	Заповнити обов’язкові поля «Email» та «Пароль», «Підтвердити пароль»	Кількість символів у полях вказана правильно	Пройдено
3	Натиснути кнопку «Зареєструватися»	Новий обліковий запис створено	Пройдено
4	Користувача перенаправлено на сторінку входу в систему	З’являється форма для входу в систему	Пройдено

Результати отримані в ході проходження цього тесту відповідають очікуваним. Помилок в роботі програмної системи виявлено не було виявлено.

Таблиця 5.2 – Тест-кейс 2 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	ТС 2		
Опис функції:	Додавання нового транспортного засобу		
Власник тесту:	Оболенцев Денис		
Дата створення:	20.06.2024		
Мета тесту:	Перевірити коректність додавання транспортного засобу		
Додавання нового транспортного засобу			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити сторінку «Транспортні Засоби»	З’являється таблиця з даними про транспортні засоби	Пройдено
2	Натиснути кнопку «Додати новий транспортний засіб»	Відкривається нове вікно з формою для заповнення даних про транспортний засіб	Пройдено
3	Заповнити обов’язкові поля «Номер авто» «Модель»	Кількість символів у полях вказана правильно	Пройдено
4	Натиснути кнопку «Додати авто»	Новий транспортний засіб додано	Пройдено
5	Користувача перенаправлено на сторінку «Транспортні Засоби»	З’являється таблиця з даними про транспортні засоби	Пройдено

Таблиця 5.3 – Тест-кейс 3 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	ТС 3		
Опис функції:	Редагування даних про автомобіль		
Власник тесту:	Оболенцев Денис		
Дата створення:	20.06.2024		
Мета тесту:	Перевірити коректність редагування даних про авто		
Редагування даних про транспортний засіб			
№	Опис випадку	Очікуваний результат	Висновок
1	Натиснути кнопку «Редагувати дані про авто»	Відкривається вікно з формою для редагування даних про авто	Пройдено
2	Змінити поля «Реєстраційний номер» «Модель», «Марка»	Кількість символів у полях вказана правильно	Пройдено
3	Натиснути кнопку «Редагувати авто»	Дані про авто змінено	Пройдено
4	Користувача перенаправлено на сторінку «Транспортні Засоби»	З’являється таблиця з даними про транспортні засоби	Пройдено

Таблиця 5.4 – Тест-кейс 4 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	ТС 4		
Опис функції:	Видалення інформації про ремонтну роботу		
Власник тесту:	Оболенцев Денис		
Дата створення:	20.06.2024		
Мета тесту:	Перевірити коректність видалення інформації про ремонтну роботу		
Видалення інформації про ремонтну роботу			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити сторінку «Ремонтні роботи»	З’являється таблиця з даними про ремонтні роботи	
2	Натиснути кнопку «Видалити»	Відкривається діалогове вікно для підтвердження видалення	Пройдено
3	Натиснути кнопку «Підтвердити»	Обрана ремонтна робота видалена	Пройдено
4	Відкрита сторінка «Ремонтні роботи»	Таблиця з даними про ремонтні роботи оновлена	Пройдено

Отже, після проведеного тестування основного функціоналу програмної системи можна зробити висновок, що результати отримані в ході тестування відповідають очікуваним програмна система у відповідності до поставлених вимог.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи бакалавра було розроблено програмну систему для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту.

В першому розділі було проведено аналіз предметної галузі – діяльності підприємства з ремонту та технічного обслуговування транспортних засобів, розглянуто існуючі програмні системи, що дозволяють здійснювати управління та облік роботи підприємства з ремонту та технічного обслуговування автотранспорту також виявлено їх переваги та недоліки, що дозволило виявити проблемні питання, які необхідно вирішити в ході проектування програмної системи та сформувати задачі. В другому розділі було сформовано вимоги до програмної системи та обрано наступні технології та інструменти для розробки програмної системи:

- для зберігання даних було обрано систему управління реляційними базами даних MySQL, що надає можливість використовувати структуровану мову запитів SQL для управління даними в базі даних;
- для розробки програмної системи було обрано мову програмування JavaScript;
- для розробки серверної частини обрано Node.js та фреймворк Express;
- клієнтську частину розроблено за допомогою JavaScript-бібліотеки React.

В третьому розділі було розроблено структуру зберігання даних та спроєктовано архітектуру програмної системи, що являє собою клієнт-серверний веб-застосунок. В четвертому розділі було проведено функціональне тестування програмної системи та виявлено відповідність отриманих результатів тестування функціональним вимогам до програмної системи.

Отже, результатом кваліфікаційної роботи є створення програмної системи для управління та обліку роботи підприємства з ремонту та технічного обслуговування автотранспорту, що дозволяє автоматизувати оброблення даних та підвищити ефективність моніторингу та управління робочими процесами підприємства.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Про затвердження Положення про технічне обслуговування і ремонт дорожніх транспортних засобів автомобільного транспорту: Наказ Міністерства транспорту України від 30.03.1998 р. № 102. URL: https://zakon.rada.gov.ua/laws/show/z0268-98#Text (дата звернення: 18.06.2024).

2. Про затвердження Правил експлуатації колісних транспортних засобів: Наказ Міністерства інфраструктури України від 26.07.2013 р. № 550. URL: https://zakon.rada.gov.ua/laws/show/z1453-13#Text (дата звернення: 18.06.2024).

3. Захарчук О.В. Технічне обслуговування і ремонт АТЗ : навч. посіб. для студ. вищих навч. закладів / Олег Вікторович Захарчук. – Луцьк : РВВ Луцького НТУ, 2015. – 140 с.

4. Марчук Р.М. Конспект лекцій з навчальної дисципліни «Технічна експлуатація автомобілів» для студентів спеціальності 274 «Автомобільний транспорт». – Рівне: НУВГП, 2017. – 140 с.

5. EasyWeek – Програма онлайн-запису та CRM управління бізнесом [Електронний ресурс]. URL: https://easyweek.com.ua/solutions/service-station (дата звернення: 20.06.2024).

6. AutoRepair Cloud – Automotive Repair Shop Software [Електронний ресурс]. URL: https://www.autorepaircloud.com (дата звернення: 20.06.2024).

7. Autobooking інтернет-портал [Електронний ресурс]. URL: https://autobooking.com/ua-ua (дата звернення: 20.06.2024).

8. С.В.Баран. Основи web-програмування: Навчальний посібник. – Кривий Ріг: Державний університет економіки і технологій, 2023. –316 с.

9. Конспект лекцій з дисципліни «Бази даних», для студентів усіх форм навчання напрямку 121 – «Програмна інженерія» [Електронне видання] / Упоряд. О.О. Мазурова. – Харків: ХНУРЕ, 2017. – 190 с.

10. Моделювання програмного забезпечення : навч.-метод. посіб. / уклад.: С. Ю. Манаков, О. Г. Трофименко, Ю. Г. Лобода, А. І. Дика ; Нац. ун-т «Одеська юрид. академія». – Одеса : Фенікс, 2023. – 145 с.

11. Серверні веб-технології [Електронний ресурс]: навч. посіб. для здобувачів ступеня бакалавра за освітньою програмою «Автоматизація та комп'ютерно-інтегровані технології кібер-енергетичних систем» спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології / КПІ ім. Ігоря Сікорського ; уклад. О. С. Бунке – Київ : КПІ ім. Ігоря Сікорського, 2023. – 109 с

12. Node.js – About Node.js. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/en/about (дата звернення: 26.06.2024).

13. Herron D. Node. js Web Development: Server-Side Web Development Made Easy with Node 14 Using Practical Examples, 5th Edition. Packt Publishing, Limited, 2020.

14. Express – це фреймворк для веб-застосунків, побудованих на Node.js. Express – Node.js web application framework. URL: https://expressjs.com/uk/ (дата звернення: 26.06.2024).

15. Sequelize Feature-rich ORM for TypeScript & JavaScript. URL: https://sequelize.org/docs/v6/ (дата звернення: 28.06.2024).

16. Quick Start – React. React. URL: https://react.dev/learn (дата звернення: 02.07.2024).

17. Починаючи Axios Docs. *Axios*. URL: https://axios-http.com/uk/docs/intro (дата звернення: 29.06.2024).

18. Авраменко А.С., Авраменко В.С., Косенюк Г.В. Тестування програмного забезпечення. Навчальний посібник. – Черкаси: ЧНУ імені Богдана Хмельницького, 2017. – 284 с.

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

StrikePlagiarism.com

Дата звіту 7/18/2024
Дата редагування ---

Звіт не був оцінений.

метадані

Заголовок
2024_Б_ПІ_ПЗПІп_22_1_Оболенцев_Д_В_скорочений

Автор
Оболенцев Денис Васильович

Науковий керівник / Експерт
Олена Володимирівна каф. ПІ Олійник

підрозділ
Харківський національний університет радіоелектроніки

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	5
Інтервали	A→	0
Мікропробіли	¶	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	40

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

9.49% 9.49% КП 1

3.05% 3.05% КЦ

25
Довжина фрази для коефіцієнта подібності 2

8978
Кількість слів

72502
Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	http://eprints.cdu.edu.ua/1482/1/testyvan.pdf	67	0.75 %
2	https://dan-it.com.ua/uk/blog/chto-jeto-takoe-node-js-prostymi-slovami/	40	0.45 %
3	http://eprints.cdu.edu.ua/1482/1/testyvan.pdf	34	0.38 %
4	тищенко-пз.docx [redacted] 6/17/2022 Odessa National Polytechnic University (Наукові журнали "ВСІТ" та "ПАІТ")	31	0.35 %
5	https://er.nau.edu.ua/bitstream/NAU/41928/1/%D0%A4%D0%9A%D0%9A%D0%9F%D0%86_2020_122_%D0%97%D0%B0%D0%B3%D1%83%D1%80%D1%81%D1%8C%D0%BA%D0%B0%D0%AE%D0%93.pdf	31	0.35 %

ДОДАТОК Б

Слайди презентації

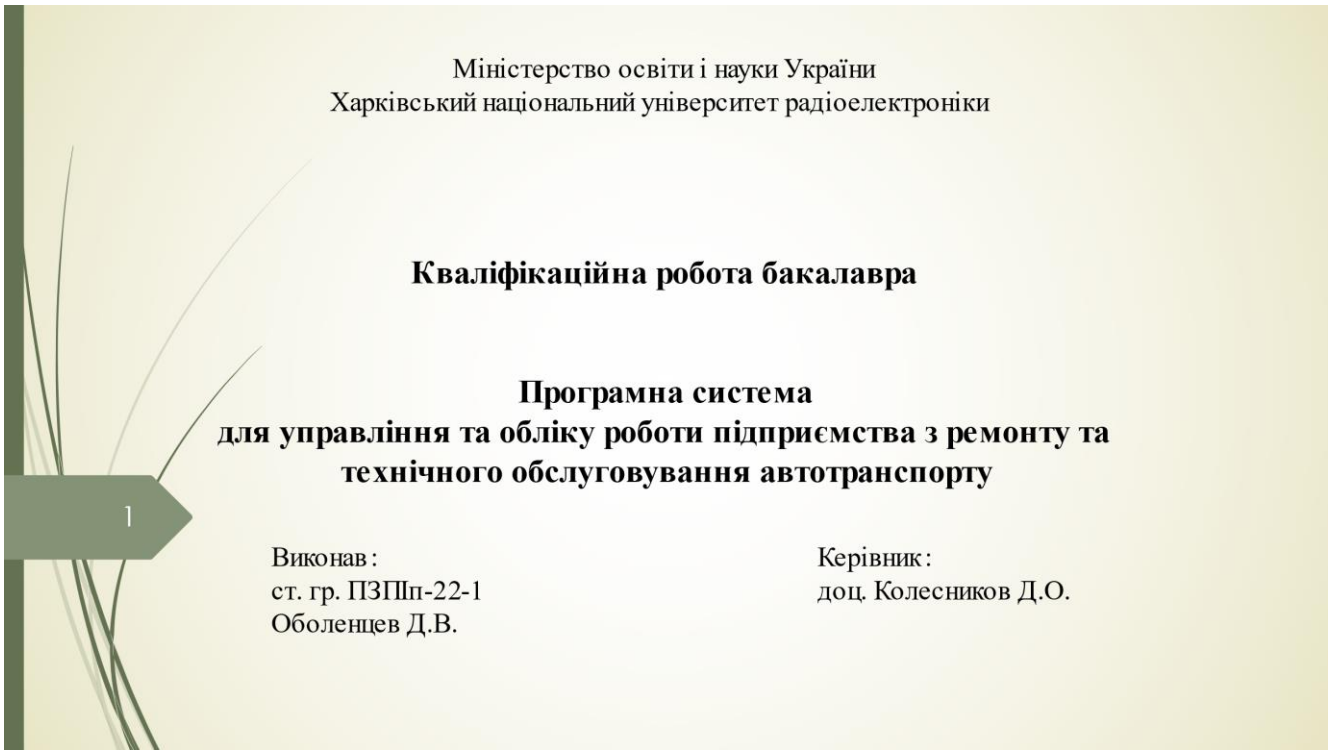


Рисунок Б.1 – Слайд 1

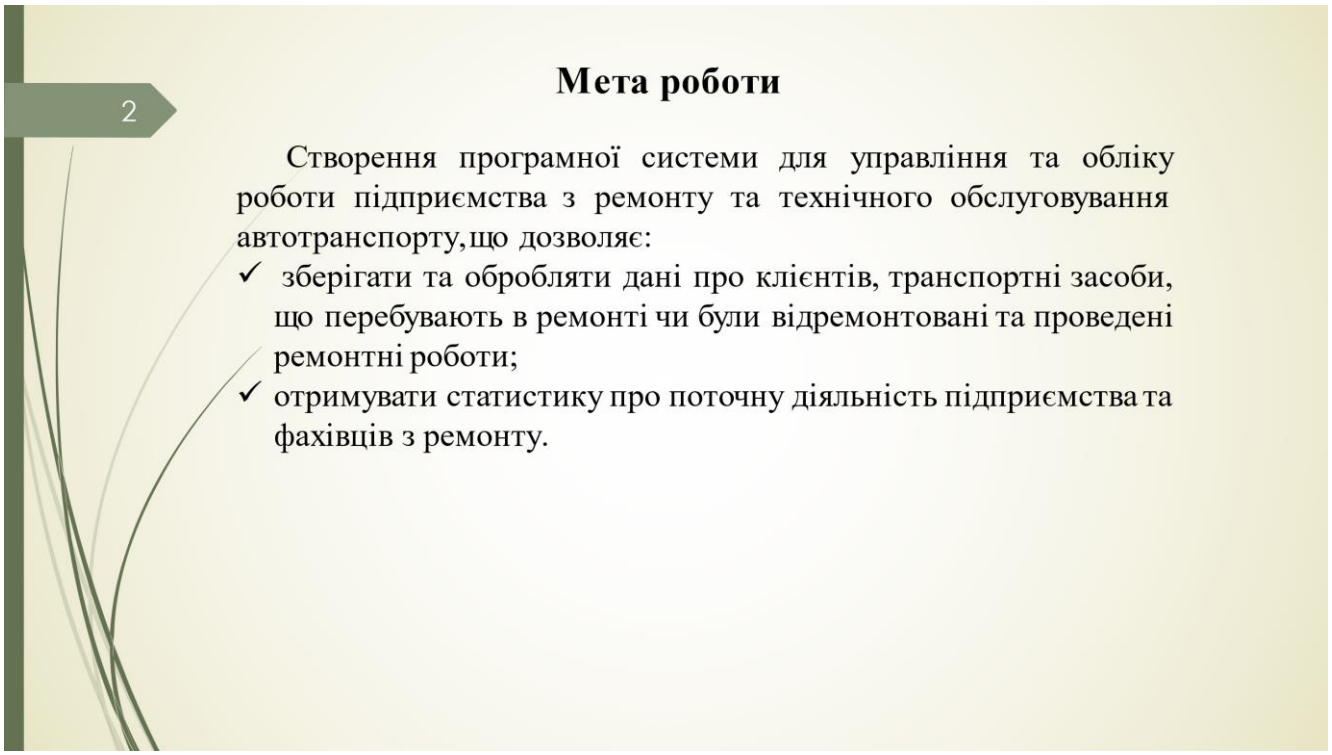


Рисунок Б.2 – Слайд 2

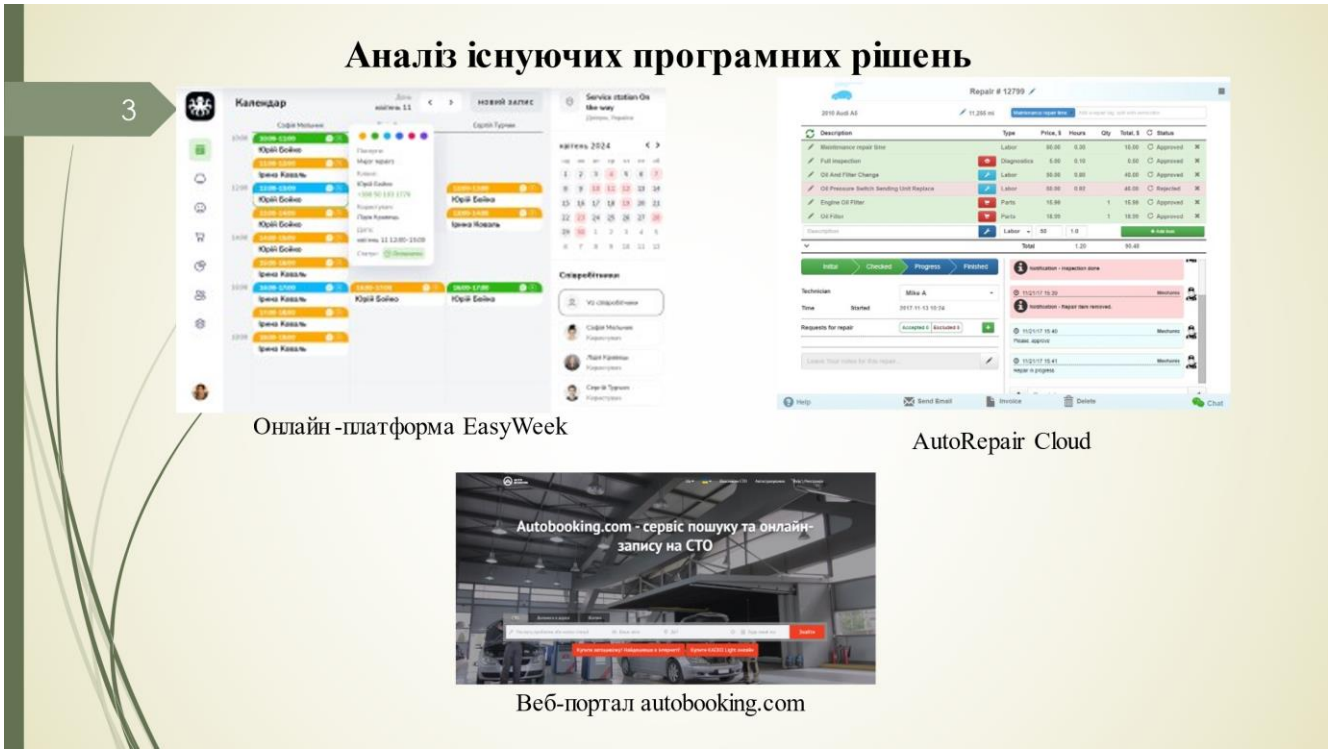
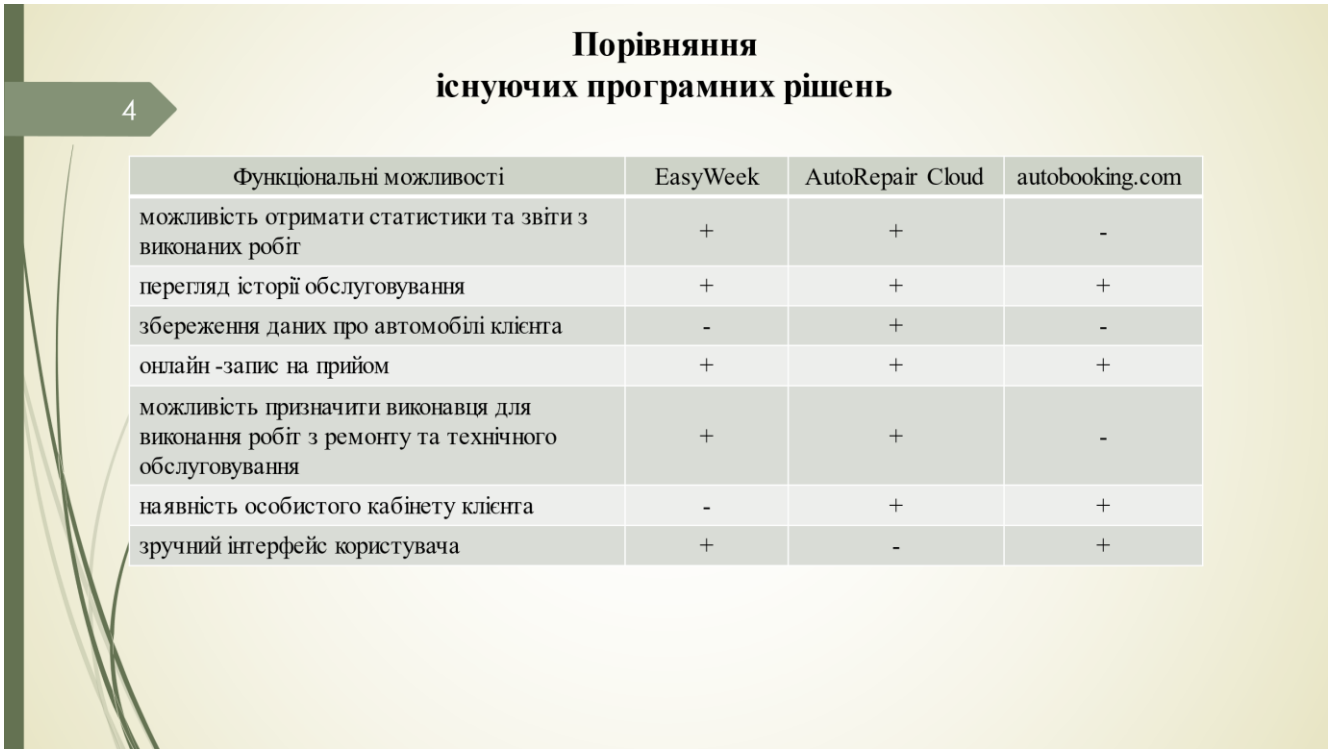


Рисунок Б.3 – Слайд 3



Функціональні можливості	EasyWeek	AutoRepair Cloud	autobooking.com
можливість отримати статистики та звіти з виконаних робіт	+	+	-
перегляд історії обслуговування	+	+	+
збереження даних про автомобілі клієнта	-	+	-
онлайн-запис на прийом	+	+	+
можливість призначити виконавця для виконання робіт з ремонту та технічного обслуговування	+	+	-
наявність особистого кабінету клієнта	-	+	+
зручний інтерфейс користувача	+	-	+

Рисунок Б.4 – Слайд 4

Постановка задачі

5

Необхідно спроектувати програмну систему, яка здійснює обробку та збереження інформації стосовно послуг, що надаються підприємством з ремонту та технічного обслуговування автотранспорту, працівників, клієнтів, їх транспортних засобів, проведених робіт з ремонту та технічного обслуговування .

Користувачі програмної системи:

- Адміністратор, здійснює контроль та управління фахівцями з ремонту та займається формуванням статистичної звітності та взаємодіє з клієнтами, здійснює реєстрацію та облік проведених ремонтних робіт;
- Інженер здійснює ремонт та технічне обслуговування автотранспорту, створює ремонтні завдання, відстежує статус виконання ремонтних робіт;
- Клієнт, здійснює онлайн-запис на прийом до інженера з ремонту.

Рисунок Б.5 – Слайд 5

USE-CASE діаграма програмної системи

6

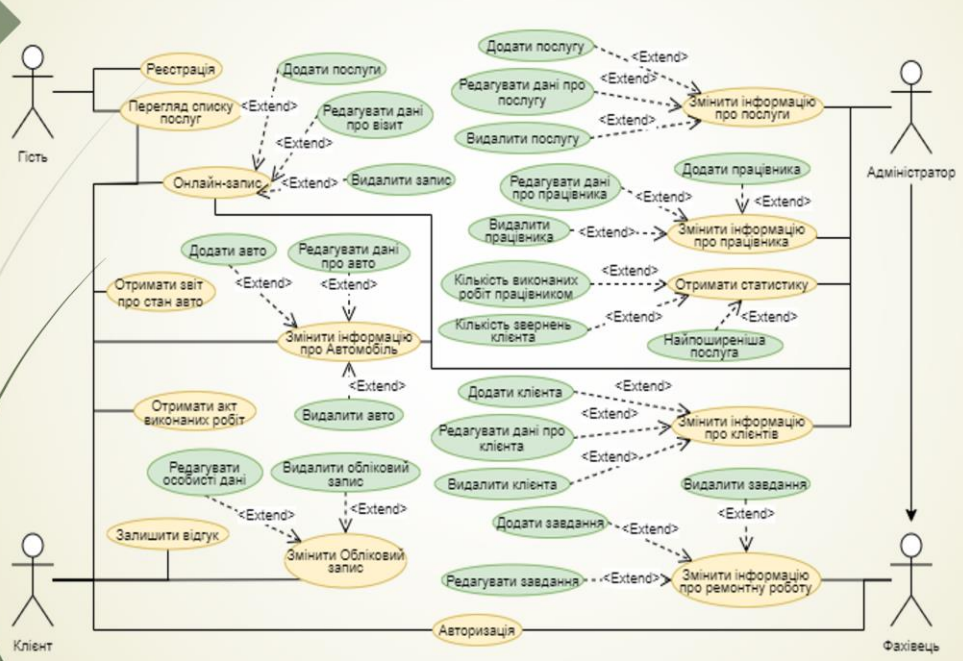


Рисунок Б.6 – Слайд 6

Рисунок Б.7 – Слайд 7

Рисунок Б.8 – Слайд 8

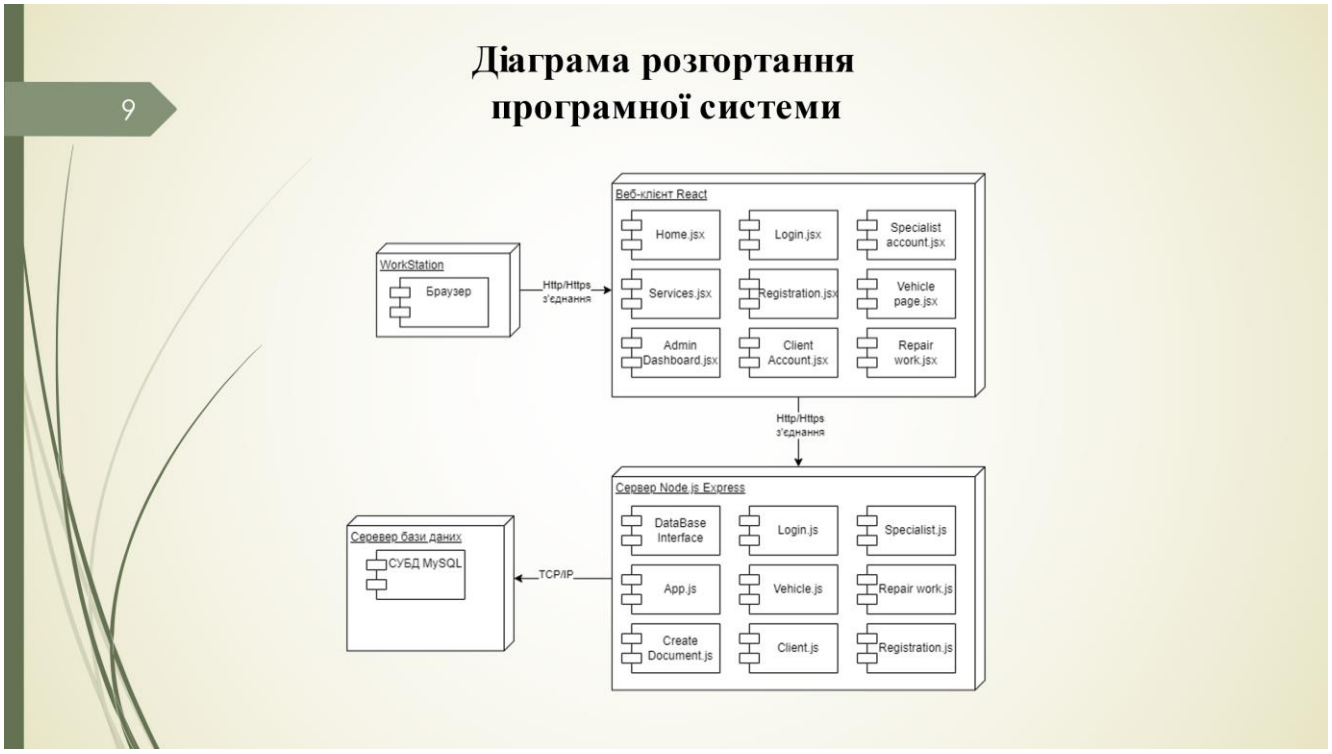


Рисунок Б.9 – Слайд 9

Рисунок Б.10 – Слайд 10

11

Приклад програмної реалізації серверної частини для таблиці «Послуги»

```
//Отримати список всіх послуг
export const  getAllServices  = async (req, res) => {
    try {
        const response = await   Service.findAll ()
        if (response.length  > 0) {
          res.status (200). json ({ success: true, message: "Services found", data: response });
        } else {
            res.status (404). json ({ success: false, message: "No services found" });
        }
    } catch (error) {
        res.status (500). json ({ success: false, message: `${error.name}: ${    error.message }` });
    }};

//Видалити послугу
export const  deleteService  = async (req, res) => {
    const id = req.params.id
    try {
        const service = await   Service.findOne ({
            attributes: ['id', 'name'],
            where: {id: req.params.id}});
        if (!service) return   res.status (404). json ({ success: false, message: 'Service not
found!' });
        await  Service.destroy ({
            where: {id: id},
            cascade: true});
        res.status (200). json ({ success: true, message: 'Deleted successfully!' })
    } catch (error) {
        res.status (500). json ({ success: false, message: 'Failed to delete!' })
    }};
```

Рисунок Б.11 – Слайд 11

12

Приклад програмної реалізації серверної частини для таблиці «Послуги»

```
//Створення нової послуги
export const  createService  = async (req, res) => {
    const { name, type, duration, price, description } =      req.body ;
    try {
        let service = null;
        service = await  Service.findOne ({
            where: {name: req.body.name}
        });
        //Перевірка чи вже було створено таку послугу
        if (service) {
            return  res.status (400). json ({ message: "Service already exists" });
        }
        service = await  Service.create ({
            name: name,  type:  type,duration : duration, price: price, description: description,
        });
        res.status (200). json ({ success: true, message: "Service was successfully created!" });
    } catch (error) {
        res.status (500). json ({ success: false, message: "Server Error!     Service was not created! Try again!" });
    }};
//Оновлення даних про послугу
export const  updateService  = async (req, res) => {
    const id = req.params.id
    try {
        const service = await   Service.findOne ({
            attributes: ['id', 'name'],
            where: {id: req.params.id}
        });
        if (!service) return   res.status (404). json ({ success: false, message: 'Service not found!' });
        await  Service.update (req.body , {where: { id: id }
        });
        res.status (200). json ({ success: true, message: 'Service was successfully updated!' })
    } catch (error) {
        res.status (500). json ({ success: false, message: 'Failed to update     !'})}
};
```

Рисунок Б.12 – Слайд 12

13

Приклад програмної реалізації серверної частини для таблиці «Послуги»

```
//Виклик функції додавання нової послуги
router.post ('/services', authenticate, restrict(['admin']),      createService );
//Виклик функції редагування послуги
router.put ('/:id', authenticate, restrict(['admin']),     updateService );
//Виклик функції видалення послуги
router.delete ('/:id', authenticate, restrict(['admin']),     deleteService );
//Виклик функції отримання списку всіх послуг
router.get ('/services/all',   getAllServices );
//Виклик функції  для пошуку  послуг
router.get ('/services/search',   findServices );

//Пошук послуги за типом назвою та ціною
export const  findServices  = async (req, res) => {
    const { query, type } =    req.query ;
    let response;
    try {const  whereConditions  = {};
        if (type) { whereConditions.type   = type;}
        if (query) {  whereConditions  [Op.or ] = [
                { name: { [ Op.like ]: `%${query}%` } },
                { price: { [ Op.like ]: `%${query}%` } },];
        }
        response = await   Service.findAll  ({ where:  whereConditions   });
        if (response.length   > 0) {
            res.status (200). json ({ success: true, message: "Found", data: response });
        } else {
            res.status (404). json ({ success: false, message: "Not found" });
        }
    } catch (error) {
        res.status (500). json ({ success: false, message: `${error.name}: ${     error.message }` });
    }
};
```

Рисунок Б.13 – Слайд 13

14

Інтерфейс користувача

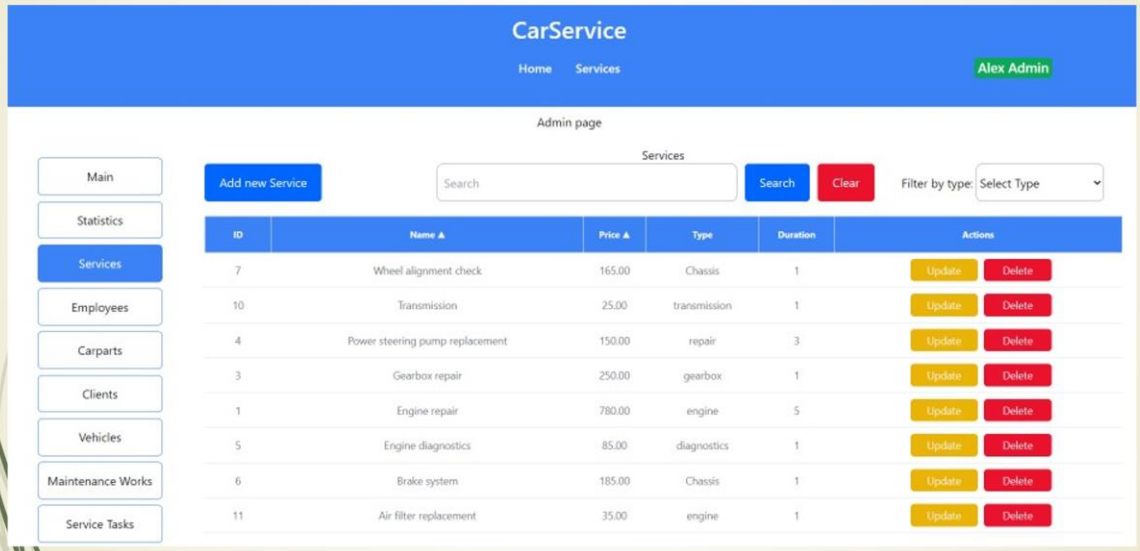


Рисунок Б.14 – Слайд 14

Тестування програмної системи

15

Тестовий випадок	Опис випадку	Вхідні дані	Очікуваний результат	Фактичний результат	Висновок
TC1	Створення нового облікового запису	Email та Пароль	Реєстрація нового клієнта	Новий обліковий запис створено	Пройдено
TC2	Додавання нового транспортного засобу	Інформація про авто: Номер, Модель,	Новий запис про авто успішно створено, відображається у списку записів	Новий запис про автомобіль додано до таблиці	Пройдено
TC3	Редагування даних про транспортний засіб	Ідентифікатор запису для видалення, Дані для редагування	Оновлення інформації про автомобіль, збереження змін в базі даних	Дані про авто змінено	Пройдено
TC4	Видалення інформації про ремонтну роботу	Ідентифікатор запису для видалення	Обраний запис видалено, таблиця з ремонтними роботами оновлена	Запис про ремонтну роботу видалено	Пройдено

Рисунок Б.15 – Слайд 15

Висновки

16

В ході виконання кваліфікаційної роботи:

- ✓ Проведено аналіз предметної галузі;
- ✓ Розглянуто існуючі програмні системи, виявлено їх переваги та недоліки, ;
- ✓ Сформовано вимоги до програмної системи;
- ✓ Проведено проєктування архітектури програмної системи;
- ✓ Розроблено програмну систему у вигляді веб-застосунку з для обліку клієнтів та ремонтних робіт, зберігання даних здійснюється за допомогою реляційної бази даних;
- ✓ Проведено тестування програмної системи.

Результатом кваліфікаційної роботи є програмна системи що дозволяє автоматизувати оброблення даних та підвищити ефективність моніторингу та управління робочими процесами підприємства з технічного обслуговування автотранспорту.

Подальшим розвитком програмної системи може бути розширення функціоналу системи наприклад, додавання сповіщення про виконання ремонтних робіт за допомогою електронної пошти.

Рисунок Б.16 – Слайд 16

ДОДАТОК В

Приклад коду програми

Файл employeeController.js (серверна частина)

```
import Employee from "../models/EmployeeModel.js";
import bcrypt from "bcryptjs";

export const createEmployee = async (req, res) => {
    const { name, surname, email, address, phone, speciality, experience,
    hireDate, password } = req.body;
    try {
        let employee = null;
        employee = await Employee.findOne({
            where: { email: req.body.email }
        });
        //check if employee exists
        if (employee) {
            return res.status(400).json({ message: "Employee already exists"
});
        }
        const salt = await bcrypt.genSalt(10);
        const hashPassword = await bcrypt.hash(password, salt);
        employee = await Employee.create({
            name: name, surname: surname,
            email: email,address: address,
            phone: phone, speciality: speciality,
            experience: experience, hire_date: hireDate,
            password: hashPassword,
        });
        res.status(200).json({ success: true, message: "Employee was
       successfully created!" });
    } catch (error) {
        res.status(500)
      .json({ success: false, message: `${error.name}: ${error.message}` });
    }
};

export const updateEmployee = async (req, res) => {
    const id = req.params.id
    try {
        const employee = await Employee.findOne({
            attributes: ['id', 'name', 'surname'],
            where: { id: req.params.id }
        });
       if (!employee) return res.status(404).json({ success: false, message:
'Employee not found!' });
        await Employee.update(req.body, {
            where: { id: id }
        });
          res.status(200).json({ success: true, message: 'Update completed
successfully!' })
    } catch (error) {
        res.status(500).json({ success: false, message: 'Failed to update!'
})
    }};
```

```
export const deleteEmployee = async (req, res) => {
    const id = req.params.id
    try {
        const employee = await Employee.findOne({
            attributes: ['id', 'name', 'surname'],
            where: {id: req.params.id}
        });
       if (!employee) return res.status(404).json({ success: false, message:
'Employee not found!' });
        await Employee.destroy({
            where: {id: id},
            cascade: true
        });
   res.status(200).json({ success: true, message: 'Deleted successfully!' })
    } catch (error) {
   res.status(500).json({ success: false, message: 'Failed to delete!' }) }
};

export const getEmployeeById = async (req, res) => {
    const id = req.params.id
    try {
        const response = await Employee.findOne({
            attributes: {
                exclude: ['password']
            },
            where: {id: id}
        });
     if (!response) return res.status(404).json({ success: false, message:
'Employee not found!' });
         res.status(200).json({ success: true, message: "Employee found",
data: response });
    } catch (error) {
     res.status(500).json({   success:   false,   message:   `${error.name}:
${error.message}` });
    }
};

export const getAllEmployees = async (req, res) => {
    try {
        const response = await Employee.findAll({
            attributes: {
                exclude: ['password']
            }
        });
        if (response.length > 0) {
           res.status(200).json({ success: true, message: "Employees found",
data: response });
        } else {
             res.status(404).json({ success: false, message: "No employees
found" });
        }
    } catch (error) {
          res.status(500).json({  success:  false,  message:  `${error.name}:
${error.message}` });
    }
};
```

Файл ServiceModel.js (серверна частина)

```
import { Sequelize } from "sequelize";
import database from "../config/database.js";

const { DataTypes } = Sequelize;
const Service = database.define(
  "services",
  {
    name: {
      type: DataTypes.STRING,
      allowNull: false,
      validate: {
        notEmpty: true,
        len: [1, 250]
      },
    },
    type: {
      type: DataTypes.STRING,
      allowNull: true,
      validate: {
        notEmpty: true,
        len: [1, 250]
      },
    },
    duration: {
      type: DataTypes.INTEGER,
      allowNull: true,
      validate: {
        notEmpty: true,
        min: 0
      },
    },
    price: {
      type: DataTypes.DECIMAL(10, 2),
      allowNull: false,
      validate: {
        notEmpty: true,
        min: 0,
      },
    },
    description: {
      type: DataTypes.STRING,
      allowNull: true,
      validate: {
        notEmpty: true,
        len: [1, 250]
      },
    },
  },
  {
    freezeTableName: true,
    timestamps: false
  }
);
export default Service;
```

Файл Services.jsx (клієнтська частина)

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import { BASE_URL, token } from '../../config.js';
import { toast } from 'react-toastify';
import { ServicesModal } from './ServicesModal.jsx';

export const Services = () => {
  const [openModal, setOpenModal] = useState(false);
  const [services, setServices] = useState([]);
  const [selectedService, setSelectedService] = useState(null);
  const [type, setType] = useState('');
  const [query, setQuery] = useState('');
  const [originalServices, setOriginalServices] = useState([]);
  const handleSearch = async () => {
    try {
    const response = await axios.get(`${BASE_URL}/services/services/search`,
{ headers: {'Authorization': `Bearer ${token}`, }, params: { query, type },
});
      setServices(response.data.data);
      toast.success(response.message)
    } catch (error) { toast.error(error.message) }
  };
  const handleClear = async => {
    setServices(originalServices);
    setQuery('');
    setType('')
  }
  const [sortPriceAsc, setSortPriceAsc] = useState(true);
  const [sortNameAsc, setSortNameAsc] = useState(true);
  const sortByName = () => {
    const sortedItems = [...services];
    sortedItems.sort((a, b) => {
      const A = a.name.toUpperCase();
      const B = b.name.toUpperCase();
      if (sortNameAsc) { return A.localeCompare(B); }
       else { return B.localeCompare(A); } });
    setServices(sortedItems);
    setSortNameAsc(!sortNameAsc);
  };
  const sortByPrice = () => {
    const sortedItems = [...services];
    sortedItems.sort((a, b) => {
      if (sortPriceAsc) { return a.price - b.price; }
       else { return b.price - a.price; }
    });
    setServices(sortedItems);
    setSortPriceAsc(!sortPriceAsc); };
  useEffect(() => {
    getServices(); }, []);

  const getServices = async () => {
    try {
      const response = await axios.get(`${BASE_URL}/services/services/all`,
{ headers: { 'Authorization': `Bearer ${token}`,   }  });
      setServices(response.data.data);
      setOriginalServices(response.data.data);
```

```
    } catch (error) {
      toast.error(error.message)
    }
  };
  //Додавання
  const addService = () => {
    setSelectedService(null);
    setOpenModal(true);
  };
  //Редагування
  const updateService = (id) => {
    setSelectedService(id); // Обрання service для редагування
    setOpenModal(true);
  };
  //Видалення
  const deleteService = async (id) => {
    const confirmDelete = window.confirm('Are you sure you want to delete?');
    try {
      //Підтвердження дії видалення
      if (!confirmDelete) {
        return;  }
      await axios.delete(`${BASE_URL}/services/${id}`, {
          headers: {'Authorization': `Bearer ${token}`, }   }   );
      getServices(); // Оновлення списку
    } catch (error) { toast.error(error.message)  }  };
  //Закриття модульного вікна
  const handleCloseModal = () => {
    setOpenModal(false);  };
  return (
    <div> <h2>Services</h2>
      <div className="grid md:grid-cols-4 grid-rows-1 gap-4"> <div>
            <button  className='btn  btn__add'  onClick={addService}>Add  new
Service</button> </div>
  <div className="col-span-2 flex justify-between gap-2">
  <input className="h-[50px] w-[400px] border border-gray-400 rounded-lg "
   type="text"  value={query}  onChange={(e) => setQuery(e.target.value)}
            placeholder="  Search" />
    <button className='btn btn__add' onClick={handleSearch}>Search</button>
                <button       className='btn       btn__delete       mr-5'
onClick={handleClear}>Clear</button>
        </div> <div>
      <label>Filter by type: </label>
      <select className="h-[50px] w-[170px] border border-gray-400 rounded-
lg " value={type} onChange={(e) => setType(e.target.value)}>
            <option value="">Select Type</option>
            <option value="diagnostics">Diagnostics</option>
            <option value="chassis">Chassis</option>
            <option value="transmission">Transmission</option>
            <option value="engine">Engine</option>
            <option value="electronics">Electronics</option>
            <option value="bodyRepair">BodyRepair</option> </select>
        </div> </div>
      {openModal && ( <ServicesModal isNew={!selectedService}
          item={selectedService} onUpdate={getServices}
          onClose={handleCloseModal}/> )}
      <table> <thead>
          <tr> <th scope="col" className="p-4">ID</th>
```

```
              <th onClick={sortByName} style={{ cursor: 'pointer' }}>Name
{sortNameAsc ? '▲' : '▼'}</th>
             <th onClick={sortByPrice} style={{ cursor: 'pointer' }}>Price
{sortPriceAsc ? '▲' : '▼'}</th>
            <th>Type</th>
            <th>Duration</th>
            <th>Actions</th> </tr>
        </thead>
        <tbody>
          {services.map((service) => (
            <tr key={service.id}>
              <td>{service.id}</td>
              <td>{service.name}</td>
              <td>{service.price}</td>
              <td>{service.type}</td>
              <td>{service.duration}</td>
              <td>
                <button className='btn btn__update h-[25px]' onClick={() =>
updateService(service)}>Update</button>
                <button className='btn btn__delete h-[25px]' onClick={() =>
deleteService(service.id)}>Delete</button>
              </td> </tr> ))}
        </tbody> </table>
    </div>)}
```

Файл Routers.jsx (клієнтська частина)

```
import React from "react";
import { Home } from "../pages/Home";
import { Services } from "../pages/Services/Services";
import { Login } from "../pages/Login";
import { Registration } from "../pages/Registration";
import { Error } from "../pages/Error";
import { Routes, Route } from "react-router-dom";
import { ClientAccount } from "../Dashboard/client-account/ClientAccount";
import { SpecialistAccount } from "../Dashboard/specialist-
account/SpecialistAccount";
import { AdminDashboard } from "../Dashboard/admin-account/AdminDashboard";
import ProtectedRoute from "./ProtectedRoute";

export const Routers = () => {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/*" element={<Error />} />
      <Route path="/home" element={<Home />} />
      <Route path="/services" element={<Services />} />
      <Route path="/login" element={<Login />} />
      <Route path="/registration" element={<Registration />} />
      <Route path="/client/profile/me" element={<ProtectedRoute
allowedRoles={['client']}> <ClientAccount /> </ProtectedRoute>} />
      <Route path="/specialist/profile/me" element={<ProtectedRoute
allowedRoles={['specialist']}> <SpecialistAccount /> </ProtectedRoute>} />
      <Route path="/admin/profile/me" element={<ProtectedRoute
allowedRoles={['admin']}> <AdminDashboard /></ProtectedRoute>} />
    </Routes>
  );};
```