НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних наук

_____ Голуб Б.Л.

"___" _____ 20 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Інформаційна система станції технічного обслуговування

Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми

Д.е.н., професор _____ Руденський Р.А.

Керівник бакалаврської кваліфікаційної роботи

Кандидат педагогічних наук, доцент ____ Волошина Тетяна Володимирівна

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Асистент кафедри інформаційних систем та технологій _____ Понзель Ярослав Юрійович

(науковий ступінь та вчене звання) (підпис)

(ПІБ)

Виконав _____ Бровчук Дмитро Іванович

(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри

Комп'ютерних наук

_____ Голуб Б.Л.

"___" _____ 20 р.

.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Бровчук Дмитро Іванович

(прізвище, ім'я, по батькові)

Спеціальність 122 – «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи Інформаційна система станції технічного обслуговування

затверджена наказом ректора НУБіП України від "16"12 2024р. №2246-С

Термін подання завершеної роботи на кафедру _25.05.2025____

Вихідні дані до бакалаврської кваліфікаційної роботи: опис програмного забезпечення

Перелік питань, які потрібно розробити:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів.
3. Проектування системи.
4. Висновки.

Дата видачі завдання "_____"_____20___ р.

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Волошина Тетяна Володимирівна

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Бровчук Дмитро Іванович

(підпис) (ПІБ студента)

Календарний план

№ з/п	Назва етапів виконання бакалаврської кваліфікаційної роботи	Строк виконання етапів бакалаврської кваліфікаційної роботи	Примітка
1	Формулювання теми та цілей кваліфікаційної роботи	23.03.2025 – 27.03.2025	
2	Аналіз предметної області та аналогічних систем	27.03.2025 – 01.04.2025	
3	Проектування архітектури інформаційної системи	01.04.2025 – 07.04.2025	
4	Розробка бази даних та бекенд-логіки	07.04.2025 – 25.04.2025	
5	Розробка інтерфейсу користувача (веб-частина)	25.04.2025 – 09.05.2025	
6	Інтеграція компонентів, налагодження	09.05.2025 – 16.05.2025	
7	Тестування, перевірка працездатності системи	16.05.2025 – 18.05.2025	

8	Написання пояснювальної записки	18.05.2025 – 23.05.2025	

Студент _____ _Бровчук Дмитро Іванович_

(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи

_____ ____Волошина Тетяна Володимирівна

(підпис) (прізвище та ініц

Вступ

Сучасна станція технічного обслуговування (СТО) обробляє значний обсяг інформації щодо клієнтів, транспортних засобів, виконаних послуг, використаних запчастин тощо. В умовах зростаючих вимог до швидкості, точності й зручності обслуговування виникає потреба в ефективній інформаційній системі, яка дозволяє автоматизувати рутинні операції, підвищити якість обслуговування клієнтів і забезпечити облік виконаних робіт.

Метою дипломної роботи є розробка веб-орієнтованої інформаційної системи обліку СТО, що дозволяє адміністратору створювати замовлення, реєструвати клієнтів, додавати автомобілі, вибирати послуги та отримувати звіти. Система має забезпечити базовий функціонал із зручним інтерфейсом, підтримкою збереження даних у базі MySQL та можливістю адміністрування.

Об'єктом дослідження є процеси інформаційного обліку на станціях технічного обслуговування.

Предметом дослідження — методи проєктування і реалізації веб-орієнтованих інформаційних систем на прикладі СТО.

Практичне значення дипломної роботи полягає в можливості подальшого впровадження системи у реальну діяльність СТО для автоматизації документообігу та оптимізації щоденних операцій.

У пояснювальній записці подано опис технологій, що використовувались, обґрунтування вибору архітектури, моделювання структури бази даних, проєктування маршрутизації, реалізацію користувацького інтерфейсу, а також оцінку ризиків і аспектів безпеки.

Перелік умовних позначень

Позначення	**Розшифрування**
СТО	Станція технічного обслуговування
ІС	Інформаційна система
UI	User Interface (користувацький інтерфейс)
UX	User Experience (досвід користувача)
CRUD	Create, Read, Update, Delete — базові операції з даними
DB	Database (база даних)
SQL	Structured Query Language — мова структурованих запитів
API	Application Programming Interface — інтерфейс прикладного програмування
JSON	JavaScript Object Notation — формат обміну даними
HTML	HyperText Markup Language — мова розмітки гіпертексту
CSS	Cascading Style Sheets — таблиці стилів каскадного типу
JS	JavaScript — мова програмування клієнтської частини
Node.js	JavaScript середовище для серверної частини
MySQL	Система керування базами даних
ERD	Entity-Relationship Diagram — діаграма сутність-зв’язок
MVC	Model-View-Controller — шаблон архітектури ПЗ

REST	Representational State Transfer — архітектурний стиль API

Таблиця 1.1. Перелік умовних позначень

1 АНАЛІТИЧНИЙ РОЗДІЛ

1.1 Актуальність теми

У сучасному світі, де темпи автоматизації бізнесу постійно зростають, питання обліку, управління та контролю за діяльністю підприємств стає особливо актуальним. Станції технічного обслуговування (СТО) не є винятком — попит на якісне та швидке обслуговування автомобілів вимагає від бізнесу впровадження сучасних інформаційних рішень. Відсутність систематизованого обліку призводить до втрати клієнтів, неефективного управління ресурсами та помилок у веденні документації.

Створення інформаційної системи обліку СТО дозволяє автоматизувати основні процеси: реєстрацію клієнтів, облік транспортних засобів, формування замовлень на послуги, контроль термінів виконання робіт, розрахунок вартості послуг тощо. Така система зменшує навантаження на персонал, покращує якість обслуговування клієнтів та підвищує загальну ефективність роботи СТО.

Крім того, розробка подібного ПЗ є чудовою практикою для майбутнього фахівця з комп’ютерних наук, оскільки вимагає знання архітектури вебсервісів, баз даних, принципів UI/UX, а також вміння організовувати взаємодію між клієнтською та серверною частинами застосунку.

Отже, тема є актуальною як з точки зору практичного використання, так і з позиції підготовки фахівців до реальних викликів ринку праці.

1.2 Тематика бакалаврських кваліфікаційних робіт

Тематика бакалаврських кваліфікаційних робіт повинна відображати актуальні завдання галузі комп'ютерних наук, бути орієнтованою на вирішення прикладних, дослідницьких або проєктних завдань і мати практичне значення для підвищення ефективності інформаційних процесів в організаціях, установах або бізнес-середовищі.

Обрана тема «Інформаційна система обліку Станції Технічного Обслуговування» повністю відповідає спеціальності 122 «Комп'ютерні науки». Вона є комплексною та охоплює широке коло компетентностей, необхідних сучасному фахівцю, серед яких:

- аналіз і моделювання предметної області;
- проєктування структури бази даних на основі вимог до зберігання та обробки інформації;
- створення архітектури вебзастосунку з поділом на клієнтську і серверну частини;
- реалізація зручного інтерфейсу для користувача з урахуванням принципів UI/UX дизайну;
- організація взаємодії між frontend і backend компонентами за допомогою RESTful API;
- реалізація CRUD-функціоналу (створення, читання, редагування, видалення);
- обробка помилок, тестування функціональності;

- захист та валідація даних;
- написання технічної та пояснювальної документації.

Тематика бакалаврської роботи формувалася з урахуванням:

- наукових і практичних інтересів кафедри комп'ютерних наук;
- поточних потреб цифровізації сфери обслуговування та обліку;
- тенденцій у веброзробці, хмарних рішеннях і збереженні даних;
- можливостей студента самостійно спроєктувати, реалізувати та протестувати ПЗ, що повністю відповідає освітній програмі.

У межах даної теми студент має змогу застосувати знання з таких дисциплін: «Системне програмування», «Бази даних», «Вебтехнології», «Об'єктно-орієнтоване програмування», «Інженерія програмного забезпечення», «Захист інформації», «Проєктування інформаційних систем» тощо.

Бакалаврська робота такого типу дозволяє вирішити прикладне завдання цифровізації облікових процесів на СТО, підвищити прозорість і контроль над замовленнями, оптимізувати внутрішню логістику та мінімізувати людський фактор при роботі з документацією.

Таким чином, обрана тема не лише відповідає навчальним цілям, але й має цінність у реальному бізнес-контексті. Вона забезпечує можливість студенту проявити себе як розробника повноцінної інформаційної системи — від аналізу до впровадження.

.

1.3 Аналіз предметної області

Предметною областю дослідження є діяльність Станції Технічного Обслуговування (СТО) — підприємства, що спеціалізується на наданні

комплексу послуг із діагностики, ремонту, технічного обслуговування, а також супутніх сервісів для легкових та вантажних автомобілів. СТО виконує як одноразові послуги, так і обслуговування на договірній основі з корпоративними клієнтами, що потребує зручного й надійного інструменту для обліку.

Управління роботою СТО охоплює низку взаємопов'язаних процесів і взаємодію між кількома категоріями суб'єктів. Для ефективного функціонування підприємства необхідно забезпечити збереження та обробку значного обсягу даних, таких як дані клієнтів, історія замовлень, облік транспортних засобів, перелік послуг, витратні матеріали тощо.

Основні суб'єкти предметної області

1. Клієнти

   Це фізичні або юридичні особи, які звертаються до СТО з метою обслуговування своїх транспортних засобів. Для кожного клієнта важливо вести історію звернень, контактні дані, прив'язані авто, заборгованості, статуси замовлень. У разі корпоративного обслуговування також актуальними є юридичні реквізити.

2. Автомобілі

   Це транспортні засоби, що належать клієнтам і є безпосередніми об'єктами діагностики чи ремонту. Для кожного авто зберігаються такі параметри, як марка, модель, номерний знак, VIN-код, дата реєстрації, пробіг тощо. Наявність точної інформації про авто дозволяє враховувати індивідуальні особливості під час надання послуг.

3. Послуги

   Це перелік операцій, які можуть надаватися СТО: від заміни мастила й шиномонтажу до повної діагностики чи ремонту підвіски. Послуги мають фіксовану назву, тривалість, ціну, а також можуть мати категорії

(електрика, двигун, кузовні роботи тощо).

4. Замовлення

   Формуються під час звернення клієнта та містять інформацію про клієнта, авто та перелік обраних послуг. До кожного замовлення прив’язується дата, статус (нове, в обробці, виконано, скасовано), підсумкова вартість і виконавець (за потреби). Ведення замовлень дозволяє контролювати навантаження працівників і своєчасне виконання послуг.

5. Адміністратор (персонал)

   Працівник, відповідальний за облік клієнтів, оформлення замовлень, контроль за станом їх виконання, формування звітів та базових аналітичних відомостей (наприклад, доходи за період або найчастіше замовлювані послуги).

Основні бізнес-процеси в межах предметної області

- Оформлення замовлення на обслуговування:

  Адміністратор створює нове замовлення, вибираючи клієнта, його авто та необхідні послуги. Система автоматично обраховує вартість і зберігає замовлення в базі.

- Облік клієнтів і транспортних засобів:

  Дозволяє уникнути дублювання даних, вести історію обслуговування для кожного клієнта чи авто, а також швидко знаходити потрібну інформацію.

- Обробка й ціноутворення послуг:

  Адміністратор обирає послуги із заздалегідь визначеного переліку. Це

дозволяє стандартизувати прайс-лист і уникнути помилок у виставленні рахунків.

- Формування звітності:
  Дає змогу керівнику переглядати фінансову та операційну інформацію: кількість замовлень за місяць, загальну суму прибутку, завантаження працівників тощо.

Необхідність автоматизації

Ручне ведення обліку в паперових журналах або у вигляді розрізнених електронних таблиць призводить до:

- втрати або дублювання даних;
- помилок при розрахунку вартості послуг;
- складнощів із формуванням звітів;
- зниження продуктивності персоналу.

Автоматизація процесів дозволяє значно підвищити швидкість обслуговування, зменшити людський фактор, підвищити прозорість і контроль, а також надати клієнтам сучасний рівень сервісу.

1.4 Аналіз існуючих аналогів

На сучасному ринку програмного забезпечення представлено низку готових рішень, призначених для обліку та автоматизації процесів на станціях технічного

обслуговування (СТО). Ці продукти мають різний рівень функціональності, складності використання, доступності та адаптивності до конкретних потреб користувача.

Огляд популярних аналогів

1. **АвтоСервіс+**

   Комерційна система, орієнтована переважно на середні та великі СТО. Пропонує широкий спектр можливостей: облік клієнтів, послуг, запчастин, робочого часу, аналітику. Однак програмне забезпечення має складний інтерфейс, потребує попереднього навчання персоналу та передбачає значні ліцензійні витрати, що не завжди виправдано для малого бізнесу.

2. **1С:Підприємство.Автосервіс**

   Це модульна система на базі 1С, що дозволяє налаштувати облік для автосервісу в рамках загальної інфраструктури підприємства. Має надлишковий функціонал для невеликих СТО, складну інтеграцію та високу вартість ліцензування. Крім того, потребує кваліфікованого адміністратора для супроводу.

3. **AutoRepair Cloud**

   Хмарна CRM-система, яка дозволяє вести облік замовлень, клієнтів і запасних частин. Основною перевагою є доступність з будь-якої точки світу. Проте система працює тільки онлайн, що створює залежність від стабільного інтернет-з’єднання та викликає труднощі при роботі в автономному режимі.

4. **Авторемонт CRM**

   Сучасний онлайн-сервіс із зручним та інтуїтивно зрозумілим інтерфейсом. Дає можливість працювати з клієнтами, формувати звіти,

керувати замовленнями. Однак система є закритою для змін — користувач не має змоги модифікувати функціонал чи структуру бази даних під власні потреби.

Основні недоліки готових рішень

Аналіз наявних систем дозволяє виділити низку спільних недоліків, які стають критичними при використанні в умовах малого або середнього бізнесу:

- **Складність адаптації під специфіку конкретного СТО.** Більшість рішень створено за універсальним шаблоном, без врахування індивідуальних процесів кожної організації.

- **Висока вартість ліцензії або підписки.** Для невеликих СТО це часто є фінансово необґрунтованим кроком.

- **Потреба в навчанні персоналу.** Наявність складного інтерфейсу або багаторівневого меню вимагає додаткового часу на освоєння.

- **Закритість вихідного коду.** Це обмежує можливість розширення функціоналу, інтеграції з іншими сервісами або змін у бізнес-логіці.

Доцільність власної розробки

Враховуючи вищенаведене, розробка власної веборієнтованої інформаційної системи обліку СТО є доцільним кроком. Вона дозволяє:

- максимально врахувати специфіку роботи конкретної організації;

- створити інтерфейс, адаптований під рівень комп’ютерної грамотності користувачів;

- реалізувати лише необхідний функціонал, без перевантаження системи зайвими модулями;

- забезпечити можливість подальшого вдосконалення, масштабування та підтримки.

Перспективи розвитку

У майбутньому планується удосконалення системи шляхом:

- додавання авторизації з правами доступу для персоналу;

- впровадження графіків завантаження майстрів та обліку запасних частин;

- реалізації повноцінного кабінету клієнта, де можна буде переглядати історію обслуговування авто;

- створення API для інтеграції з мобільним додатком або зовнішніми сервісами.

Таким чином, запропоноване рішення має потенціал розвитку в повноцінну інформаційну платформу для керування всіма аспектами діяльності СТО.

1.5 Висновки до розділу

У результаті аналітичного огляду було встановлено:

- Інформаційна система для СТО — це актуальне рішення, яке дозволяє оптимізувати бізнес-процеси.

- Тематика роботи охоплює широкий спектр навичок і знань, важливих для майбутнього спеціаліста.

- Предметна область добре формалізується, має чітку структуру та процеси.

- Аналіз аналогів показав наявність комерційних рішень, які мають суттєві недоліки або не відповідають вимогам гнучкості, доступності, простоти.

- Необхідна розробка нової адаптивної веборієнтованої інформаційної системи, що дозволить ефективно автоматизувати діяльність СТО.

2 ПРОЄКТНИЙ РОЗДІЛ

2.1. Загальна характеристика системи

[10] Інформаційна система обліку Станції Технічного Обслуговування (СТО) — це веборієнтований застосунок, розроблений з метою автоматизації ключових бізнес-процесів, пов’язаних з обслуговуванням клієнтів, обліком транспортних засобів, управлінням переліком послуг та формуванням замовлень. Основна мета

впровадження цієї системи полягає у зменшенні обсягу рутинної роботи, підвищенні точності облікових операцій та покращенні загальної ефективності управління СТО.

Система розроблена як сучасний клієнт-серверний вебзастосунок, що дозволяє працювати без потреби в додатковому програмному забезпеченні на стороні користувача. Уся робота здійснюється через веббраузер, що забезпечує зручність і гнучкість доступу.

Ключові функціональні можливості

Система надає адміністратору широкий спектр дій, необхідних для повсякденної роботи:

- Робота з клієнтами:
    - додавання нового клієнта;
    - редагування даних;
    - перегляд контактної інформації;
    - зв’язування клієнта з його транспортними засобами.
- Облік автомобілів:
    - додавання авто з вказанням марки, моделі, номерного знаку;
    - закріплення авто за конкретним клієнтом;
    - перегляд зареєстрованих автомобілів.
- Управління послугами:
    - вибір однієї або кількох послуг із попередньо заданого списку;
    - відображення вартості та тривалості кожної послуги;
    - можливість подальшого оновлення переліку послуг.
- Формування замовлень:
    - створення замовлення шляхом поєднання клієнта, авто та вибраних послуг;
    - автоматичний розрахунок підсумкової вартості;

- призначення статусу замовлення;
- перегляд історії замовлень.

- Додаткові можливості:
  - авторизація адміністратора через форму входу;
  - контроль доступу до функціоналу;
  - можливість видалення помилково створених замовлень;
  - візуальний зворотний зв’язок при взаємодії (повідомлення про успіх або помилку).

Технічна реалізація

З точки зору архітектури, система реалізована за класичною моделлю "клієнт-сервер", де фронтенд відповідає за інтерфейс, а бекенд — за логіку, обробку даних і взаємодію з базою даних.

Корігають інформацію про клієнтів, авто, послуги, замовлення та асоціативні зв’язки між ними.
Застосовується реляційна модель даних, яка забезпечує цілісність та зв’язність між сутностями.

Принципи роботи системи

Система реалізує повний набір операцій CRUD:

- Create (створення): додавання нових клієнтів, авто, послуг і замовлень;
- Read (читання): перегляд інформації, списків і деталей записів;
- Update (оновлення): редагування клієнтів, авто, послуг;
- Delete (видалення): можливість видалення записів із бази.

Особливості розгортання

Інформаційна система не вимагає встановлення спеціального ПЗ на клієнтські пристрої. Її можна розгорнути:

- на локальному сервері (наприклад, на підприємстві з внутрішньою мережею);
- на хмарному хостингу — для доступу з будь-якого пристрою в мережі.

Це дозволяє адаптувати систему під різні організаційні моделі СТО.

Користувацька взаємодія

Користувачем системи є адміністратор, що працює через зрозумілий і мінімалістичний вебінтерфейс. Усі дії супроводжуються підказками, повідомленнями про помилки або успіх, а навігація реалізована через зручне горизонтальне меню. Інтерфейс побудований із використанням форм введення, списків, селекторів і кнопок, що робить роботу з системою швидкою й інтуїтивною.

2.2 Архітектура інформаційної системи

Архітектура створеної інформаційної системи обліку СТО побудована за моделлю "клієнт-сервер", що є типовим рішенням для веборієнтованих застосунків. Така структура дозволяє розділити логіку системи на два основні рівні:

- Клієнтський рівень — інтерфейс користувача, реалізований за допомогою HTML, CSS та JavaScript. Він відповідає за візуалізацію, обробку взаємодії з користувачем та формування HTTP-запитів до сервера.

- Серверний рівень — вебсервер на базі Node.js (з фреймворком Express), який реалізує бізнес-логіку, обробляє запити, взаємодіє з базою даних MySQL і повертає відповіді у форматі JSON.

Така архітектура забезпечує масштабованість, гнучкість, можливість повторного використання коду й легкість супроводу системи.

Основні компоненти архітектури

- Клієнт - Інтерфейс адміністратора СТО, що працює у браузері
- Сервер - Node.js + Express, приймає запити, взаємодіє з БД, повертає відповіді
- База даних (MySQL) - Реляційна БД, у якій зберігаються всі дані: клієнти, авто, послуги, замовлення

Взаємодія між компонентами

1. Користувач у браузері відкриває одну зі сторінок інтерфейсу (наприклад, index.html).

2. Клієнтський скрипт надсилає запит до API-сервера для отримання або зміни даних (наприклад, GET /api/clients).

3. Сервер обробляє запит, звертається до бази даних MySQL, отримує або змінює інформацію.

4. Сервер повертає результат у форматі JSON.

5. Клієнтська частина отримує відповідь і відображає її у вебінтерфейсі.

Діаграма архітектури взаємодії клієнт-сервер-БД:

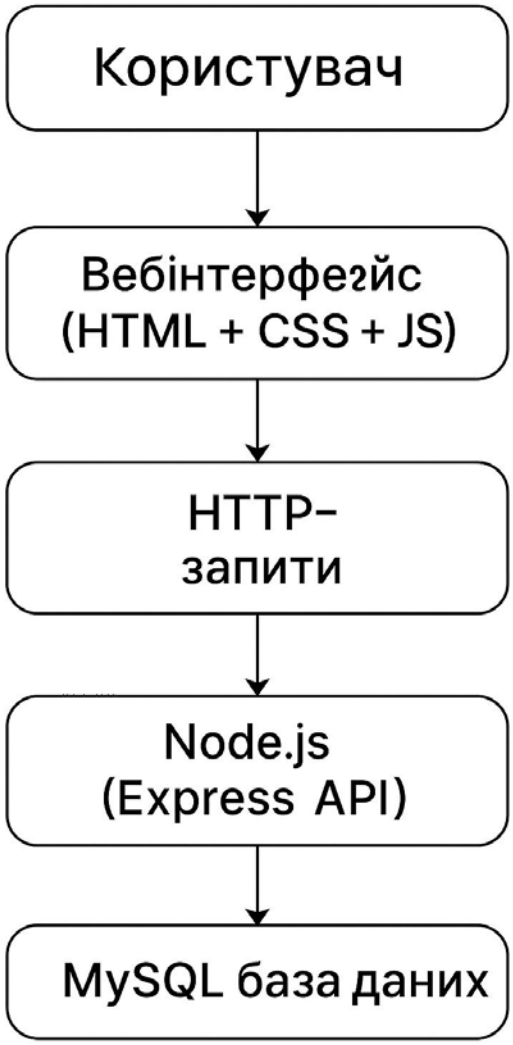


Рис. 1. Схема архітектури

2.3 Структура та проєктування бази даних

База даних є ключовим компонентом інформаційної системи обліку СТО, оскільки забезпечує зберігання, обробку та доступ до інформації про клієнтів, автомобілі, послуги та замовлення. Для зручності обслуговування та

масштабування системи було обрано реляційну модель бази даних на основі **MySQL**.

Основні вимоги до бази даних:

- підтримка зв’язків між сутностями;
- забезпечення цілісності та узгодженості даних;
- забезпечення швидкого доступу до інформації;
- можливість масштабування в майбутньому;
- зручність виконання SQL-запитів для формування звітності.

Основні сутності та зв’язки між ними

Проаналізувавши функціональні вимоги, було виокремлено основні сутності:

- **Клієнти (clients)** – зберігає інформацію про користувачів, які звертаються до СТО.
- **Автомобілі (vehicles)** – містить дані про транспортні засоби клієнтів.
- **Послуги (services)** – перелік послуг, які надаються СТО (з ціною та тривалістю).
- **Замовлення (orders)** – основна таблиця для фіксації обслуговувань.
- **Зв’язуюча таблиця orders_services** – забезпечує зв’язок «багато до багатьох» між замовленнями і послугами.

Об’єкт	**Опис**
Клієнт	Особа, яка замовляє послуги (ім’я, телефон, email)
Автомобіль	Об’єкт обслуговування (марка, модель, номер)
Послуга	Сервіс, що виконується (назва, тривалість, ціна)
Замовлення	Сукупність клієнта, авто і вибраних послуг

Опис основних таблиць

clients

- id – первинний ключ
- name – ім’я клієнта
- phone, email – контактна інформація

vehicles

- id – первинний ключ
- client_id – зовнішній ключ до таблиці clients

- brand, model, plate_number – характеристики авто

services

- id – первинний ключ
- name, price, duration – назва, вартість і тривалість послуги

orders

- id – первинний ключ
- vehicle_id – зовнішній ключ до таблиці vehicles
- date, status, total_cost – дата створення, статус, загальна вартість

orders_services

- order_id – зовнішній ключ до orders
- service_id – зовнішній ключ до services
- quantity – кількість разів, яку виконували цю послугу в межах одного замовлення

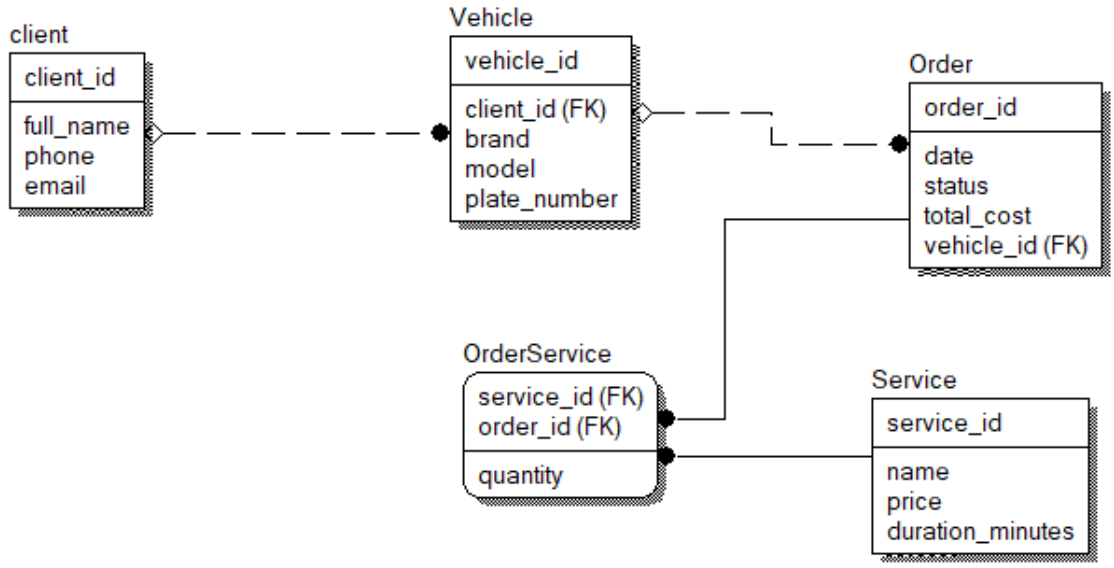


Рис.2 ER-діаграма структури бази даних

2.4 Проєктування інтерфейсу користувача

[10]Інтерфейс користувача є ключовою складовою будь-якої інформаційної системи, адже саме через нього здійснюється основна взаємодія між

користувачем та функціональністю програмного забезпечення. Якісно спроєктований інтерфейс не лише забезпечує комфортну та зрозумілу роботу з системою, але й підвищує загальну ефективність бізнес-процесів, зменшуючи час навчання персоналу та кількість помилок при роботі.

У процесі розробки інформаційної системи обліку СТО особлива увага приділялася саме ергономіці, зручності й узгодженості візуальних компонентів. Була реалізована однорідна система сторінок, які поєднані спільним навігаційним меню, що дозволяє перемикатися між основними модулями без зайвих переходів.

Основні принципи, покладені в основу інтерфейсу:

- Мінімізація кількості кліків для виконання базових дій (наприклад, додавання клієнта або створення замовлення потребує лише кількох натискань);

- Інтуїтивно зрозуміле управління — всі елементи оформлені відповідно до загальноприйнятих UI/UX-стандартів;

- Візуальна послідовність і єдність — всі форми, таблиці, кнопки мають однаковий стиль, розміри полів та інтервали;

- Сенситивність до помилок користувача — дані перевіряються до надсилання, кнопки неактивні без заповнення обов’язкових полів;

- Адаптивність — інтерфейс коректно працює на екранах різної ширини (в тому числі — на планшетах та мобільних пристроях);

- Контекстні повідомлення — при виконанні дій (створення, помилка, видалення) користувач бачить зворотний зв’язок (елементи alert, toast).

Перелік основних інтерфейсних модулів:

1. Сторінка авторизації (login.html)
   Реалізує вхід до системи лише для адміністратора. Має поле для логіна, пароля та кнопку входу. Якщо авторизація успішна, виконується перенаправлення на головну сторінку. Інакше — виводиться повідомлення про помилку. Всі сторінки перевіряють статус входу через localStorage, що унеможливлює несанкціонований доступ.

2. Головна сторінка клієнтів (index.html)
   Дає змогу створювати нових клієнтів, переглядати список існуючих у вигляді таблиці з кнопками редагування/видалення. Має форму із валідацією введених даних.

3. Сторінка додавання авто (add_vehicle.html)
   Містить випадаючий список клієнтів, що вже є в системі, форму додавання авто та підтвердження успішності створення. Всі додані авто зв’язані з відповідним клієнтом.

4. Сторінка послуг (services.html)
   Виводить перелік усіх доступних послуг із зазначенням вартості та тривалості. Кожен запис оформлено в окремий блок із тінню (box-shadow). Реалізовано перемикач темної теми.

5. Сторінка створення замовлення (orders.html)
   Дає змогу обрати клієнта, його авто, одну або кілька послуг. Сума замовлення підраховується автоматично. Форма створення має зрозумілу структуру, перевірку на коректність вибору всіх полів.

6. Сторінка перегляду замовлень (order_list.html)
   Містить таблицю з усіма замовленнями. В кожному рядку — ПІБ

клієнта, марка і модель авто, перелік послуг, дата, статус, загальна вартість, а також кнопка видалення. Дані завантажуються через API у форматі JSON.

Стильове оформлення:

Вся система оформлена в єдиній кольоровій гамі — синій, білий, світло-сірий, які асоціюються з надійністю, технічною точністю та сучасністю. Додатково реалізовано перемикач світлої/темної теми, що дає користувачеві вибір комфортного режиму роботи.

Технічні засоби реалізації інтерфейсу:

- HTML5 — розмітка сторінок;
- CSS3 — стилізація, адаптивність, ефекти;
- JavaScript — обробка подій, інтеграція з API, маніпуляції DOM;
- localStorage — збереження інформації про вхід та тему;
- Fetch API — взаємодія з backend.

Візуальна демонстрація:

У додатках до цієї пояснювальної записки подано скріншоти всіх основних сторінок: авторизація, клієнти, авто, послуги, створення замовлення, список замовлень. Для кожного представлено реальний приклад із заповненими даними, що дозволяє оцінити повноцінність реалізації системи.

Нижче представлені скріншоти сторінок веб-сайту.

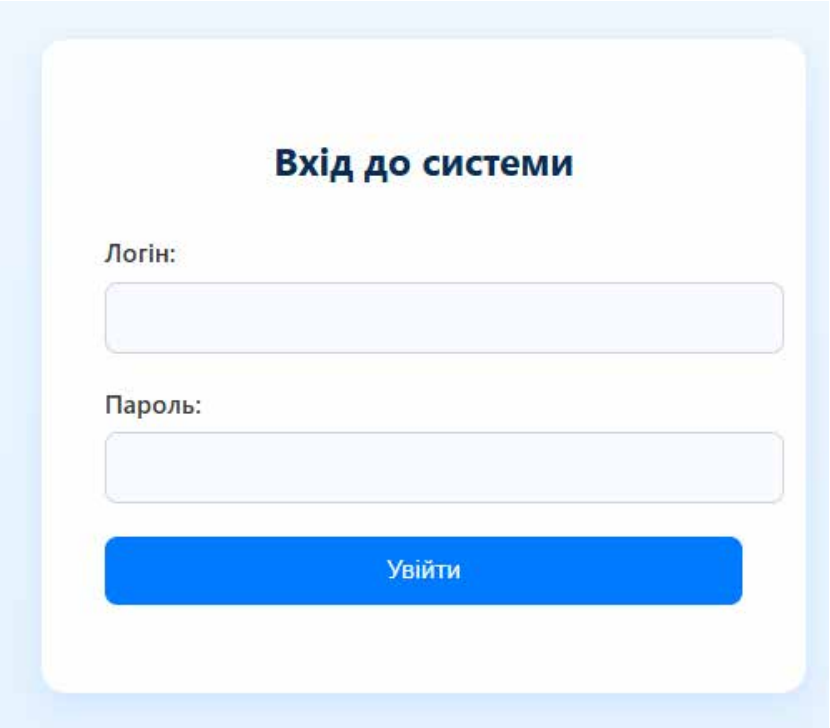


Рис.3 Сторінка Авторизації

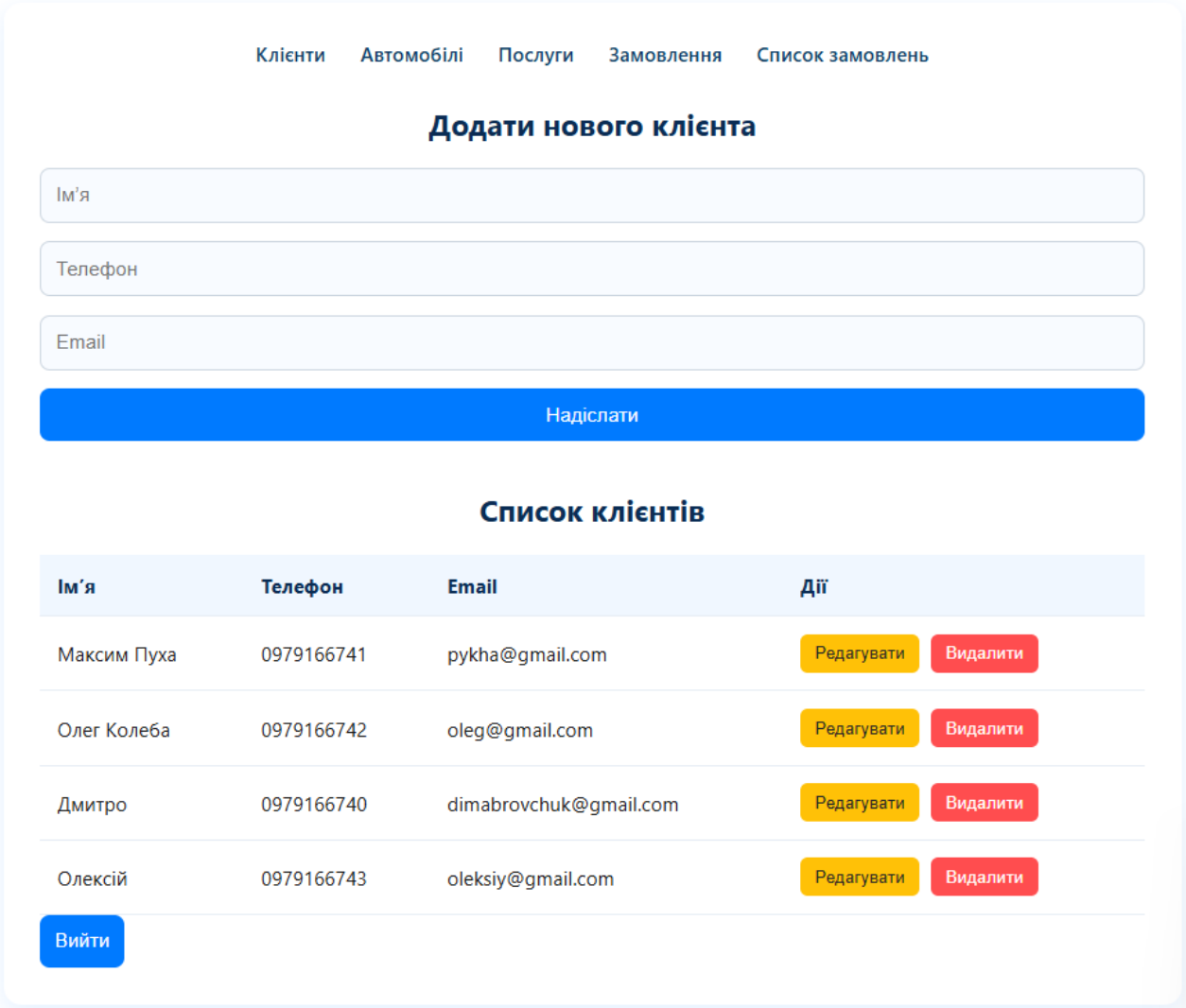


Рис. 4 Сторінка клієнтів

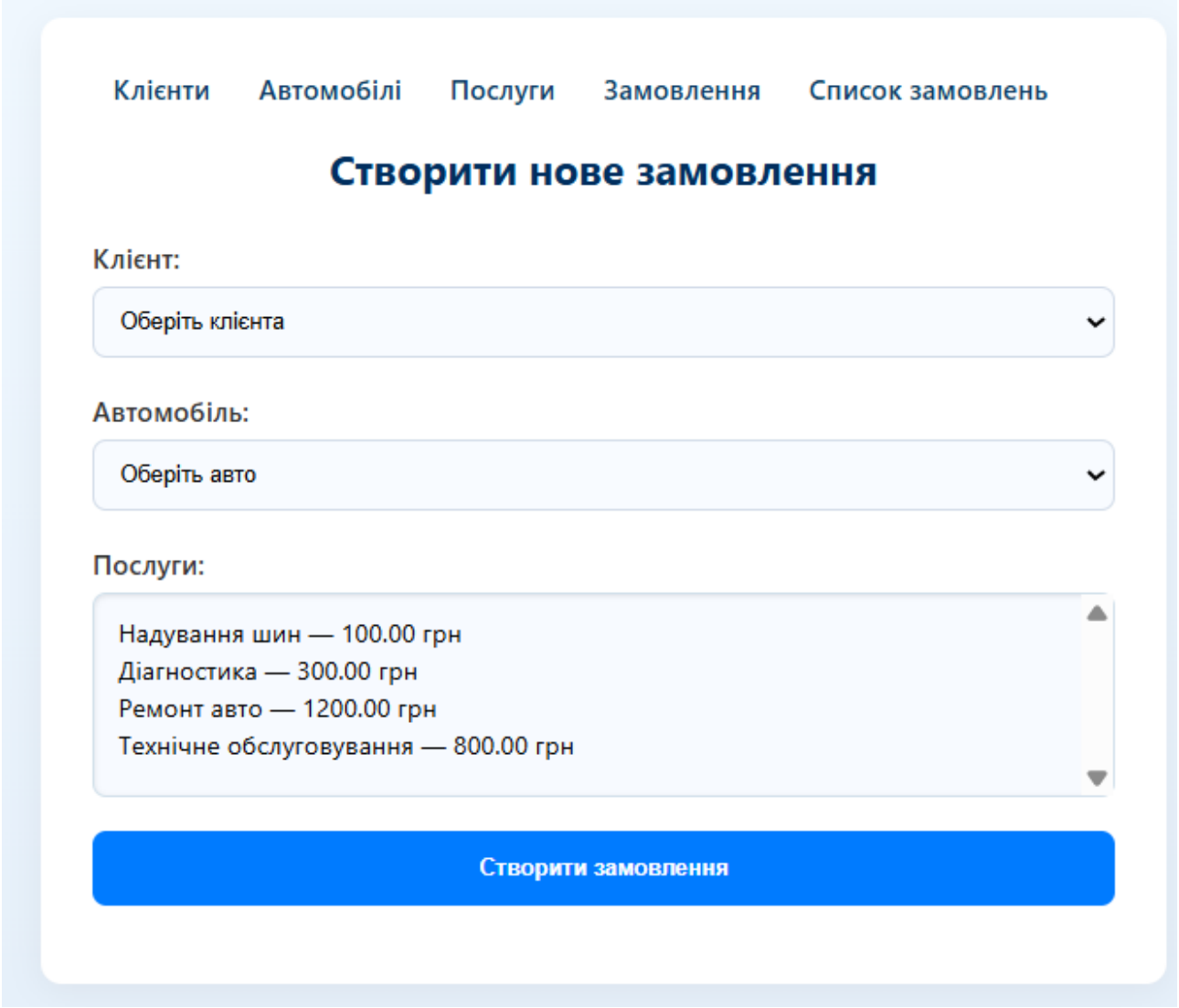


Рис.5 Сторінка формування замовлення

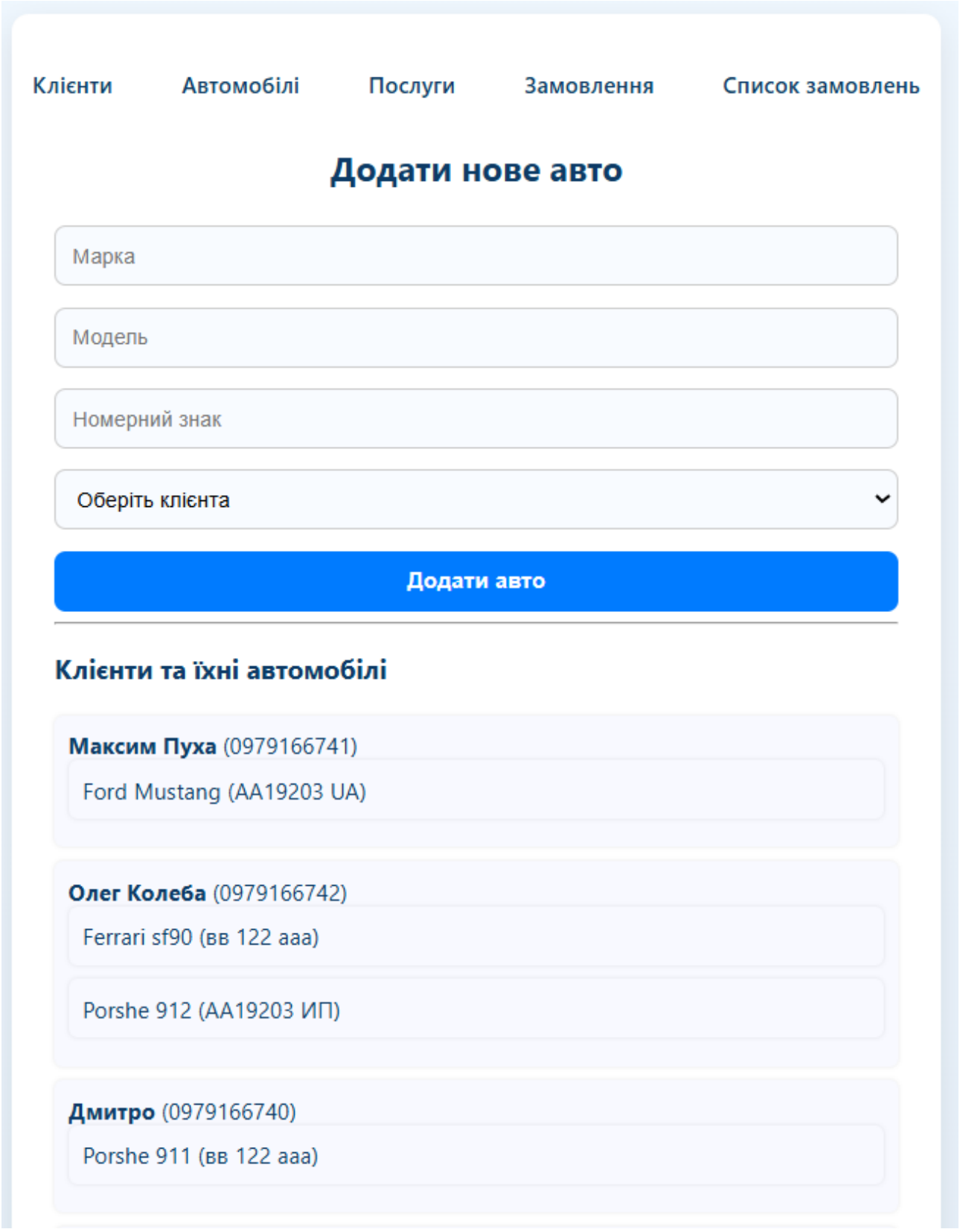


Рис.6 Сторінка Додавання Авто

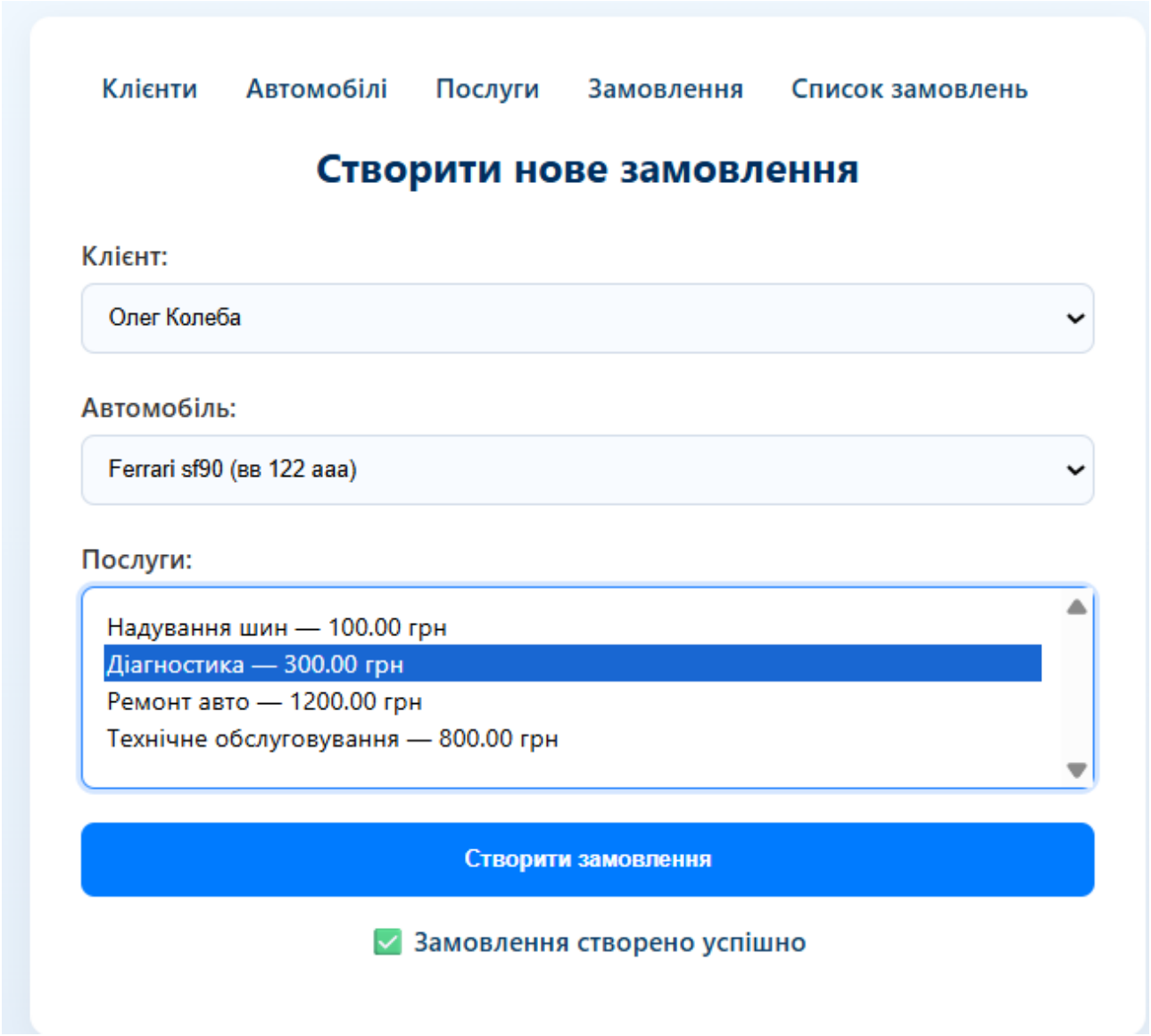


рис. 7 Приклад створення замовлення

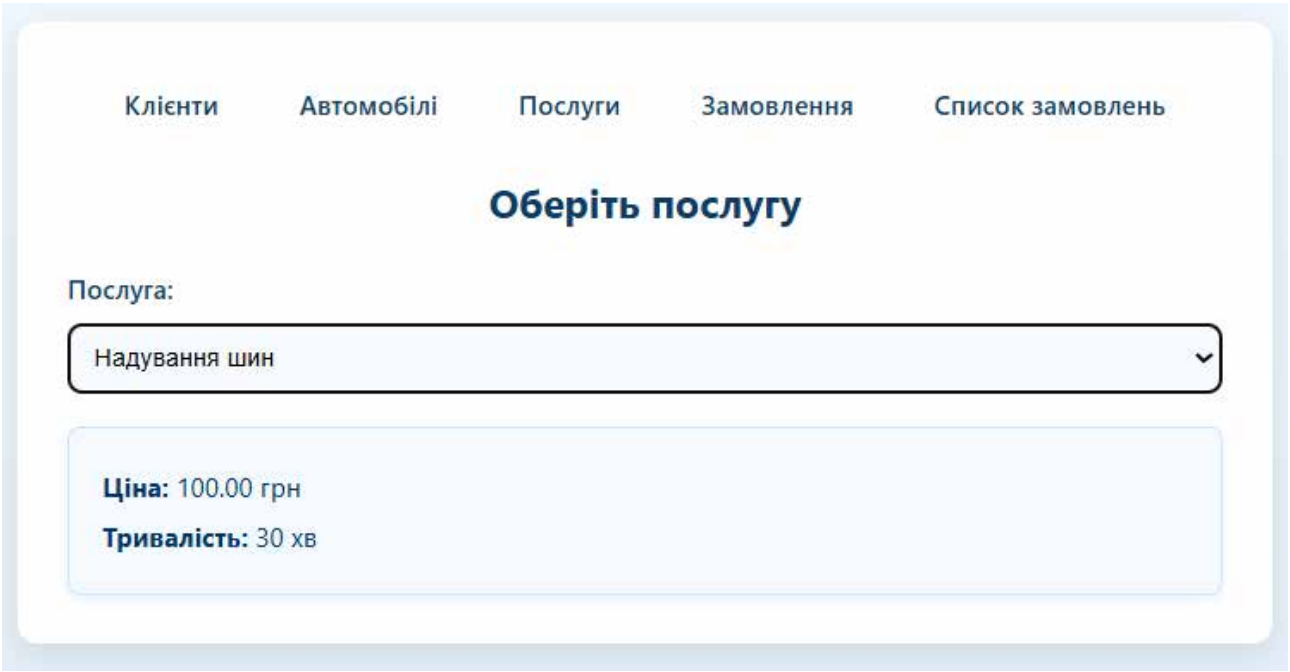


Рис.8 Сторінка перегляду послуг

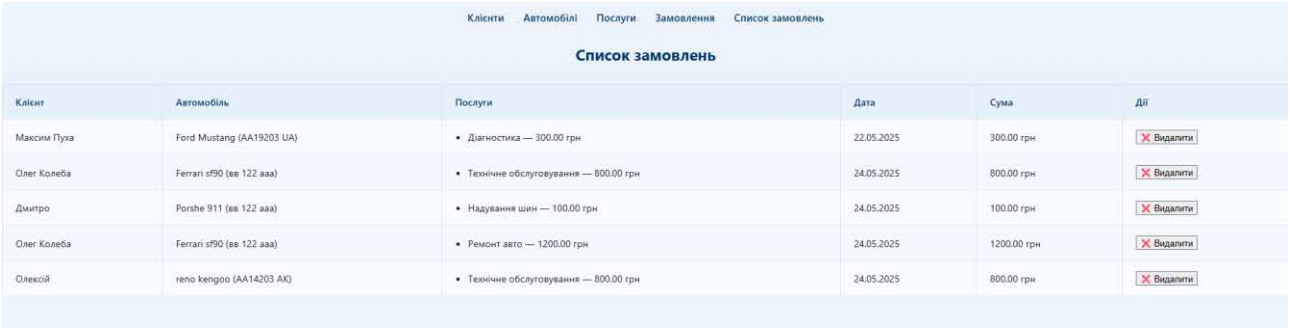


Рис.8 Сторінка перегляду замовлень

Додаток А

2.5 Реалізація функціоналу системи

У процесі розробки інформаційної системи обліку СТО реалізовано низку основних функціональних можливостей, що забезпечують повноцінну роботу сервісу. Усі функції були реалізовані на базі технологій Node.js, Express.js, MySQL, а також HTML, CSS, JavaScript на клієнтській частині.

Реалізовані функції:

1. Додавання клієнтів. На головній сторінці системи реалізована форма додавання нового клієнта, яка містить поля: ім’я, номер телефону та електронна адреса. Дані надсилаються POST-запитом на сервер та зберігаються в базі даних. Також реалізовано відображення списку клієнтів з можливістю редагування та видалення.

2. Додавання автомобіля клієнта. Реалізовано форму введення марки, моделі, номерного знаку автомобіля, а також вибору клієнта, якому належить авто. Автомобілі зв’язані з клієнтами через зовнішній ключ (client_id), що дозволяє зберігати інформацію про власників.

3. Додавання та перегляд послуг. У системі створено форму додавання та вибору послуг із зазначенням вартості та тривалості кожної. Послуги зберігаються у

відповідній таблиці бази даних та доступні для вибору під час створення замовлення.

4. Створення замовлення. На сторінці створення замовлення користувач обирає клієнта, його автомобіль та одну або кілька послуг. В результаті надсилається запит на сервер, де відбувається створення запису в таблиці orders з підрахунком загальної вартості. Всі обрані послуги записуються в таблицю зв'язку OrderService.

5. Перегляд усіх замовлень. На сторінці order_list.html реалізовано виведення повної таблиці замовлень з інформацією про клієнта, автомобіль, перелік послуг, дату та суму. Таблиця будується динамічно на основі відповіді сервера.

6. Видалення замовлення. Кожне замовлення має кнопку “Видалити”, яка надсилає DELETE-запит на сервер та видаляє відповідний запис з бази даних. Також видаляються пов’язані записи в таблиці OrderService.

7. Авторизація адміністратора. Реалізовано базову форму входу до системи з перевіркою логіна і пароля. Доступ до функціоналу доступний лише після авторизації. Авторизація реалізована на основі логіки перевірки введених облікових даних (admin / passwordmanager).

8. Вивід інформації про послугу. На сторінці перегляду послуги при виборі зі списку динамічно підтягується інформація про вартість і тривалість.

2.6 Висновки до розділу

У цьому розділі було виконано повний цикл проєктування інформаційної системи обліку СТО, що включає аналіз вимог, побудову архітектурної моделі, розробку структури бази даних, інтерфейсів користувача та реалізацію функціональних можливостей системи.

Розроблена система дозволяє ефективно керувати даними про клієнтів, автомобілі, послуги та замовлення. Архітектура клієнт-сервер забезпечує розподілення логіки між фронтендом і бекендом, що підвищує масштабованість та гнучкість у розгортанні системи.

База даних має чітко продуману структуру з реалізацією зв’язків між таблицями, що забезпечує цілісність і узгодженість збережених даних. Інтерфейс користувача реалізовано з урахуванням принципів зручності, простоти та доступності, що забезпечує комфортну роботу кінцевого користувача.

Таким чином, завершено проєктування системи, яке лягло в основу подальшої реалізації, тестування та впровадження програмного продукту в рамках дипломної роботи.

3 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

3.1 Середовище розробки

Для реалізації інформаційної системи обліку СТО було обрано набір сучасних, перевірених у промисловій розробці інструментів, що дозволяють ефективно

реалізовувати як клієнтську, так і серверну частину вебзастосунку. Вибір кожного компонента середовища обґрунтовувався його функціональними можливостями, популярністю у спільноті розробників, підтримкою документації та зручністю інтеграції в єдиний технологічний стек.

Основним середовищем розробки обрано Visual Studio Code (VS Code) — легкий, швидкий та кросплатформний редактор коду, що підтримує широкий спектр мов програмування, інтегрується з системами контролю версій, має велику кількість розширень (плагінів), а також можливість зручного налагодження коду (debugging) і форматування. У рамках проєкту використовувалися такі плагіни, як *Prettier* для автоматичного форматування, *ESLint* для перевірки синтаксису, а також плагіни для роботи з Node.js та SQL.

Серверну частину інформаційної системи реалізовано за допомогою Node.js (версія 18.x) — високопродуктивного середовища виконання JavaScript-коду на стороні сервера, що дозволяє створювати масштабовані мережеві застосунки. Node.js забезпечує неблокуючу, асинхронну модель обробки запитів, що особливо корисно для систем, які активно взаємодіють із базою даних.

У парі з Node.js було використано Express.js — легкий та гнучкий фреймворк, який значно спрощує створення RESTful API, управління маршрутами, middleware-функціями та взаємодією з клієнтом. Express.js надає зручний інструментарій для обробки HTTP-запитів і відповідає сучасним архітектурним принципам побудови серверних застосунків.

Для проєктування, адміністрування та візуалізації структури реляційної бази даних використовувалося MySQL Workbench. Цей інструмент дозволяє створювати ER-діаграми, будувати складні SQL-запити, виконувати міграції бази даних та моніторити продуктивність. База даних у проєкті використовується для зберігання інформації про клієнтів, транспортні засоби, послуги, замовлення та інші ключові об'єкти доменної області.

Тестування клієнтської частини застосунку здійснювалося в браузері Google Chrome з активним використанням DevTools — вбудованого інструментарію для налагодження, профілювання продуктивності, перевірки DOM-структури, аналізу мережевих запитів, відстеження помилок JavaScript та адаптивного перегляду інтерфейсу на різних екранах.

Для перевірки запитів до API, налагодження логіки взаємодії між фронтендом і сервером, а також тестування авторизації, маршрутизації та обробки даних використовувався Postman — потужний інструмент для REST-клієнтів. У процесі розробки створювалися окремі колекції запитів для основних груп ресурсів: клієнти, авто, послуги, замовлення.

Система контролю версій Git використовувалася для відстеження змін у кодовій базі, створення комітів, гілок, а також злиття функціоналу з різних гілок у основну. Це забезпечило надійне збереження історії змін, можливість повернення до стабільних версій та спільну роботу з проєктом.

Клієнтська частина розроблялася з використанням стандартних вебтехнологій:

- HTML5 — для створення логічної структури вебсторінок, форм, навігації та розміщення контенту;
- CSS3 — для оформлення зовнішнього вигляду інтерфейсу, реалізації адаптивного дизайну, анімацій та медіа-запитів;
- Vanilla JavaScript — як базовий інструмент для реалізації динамічної поведінки сторінок, взаємодії з DOM-елементами, обробки подій та відправлення запитів до API.

Розробка здійснювалась в операційній системі Windows 10. Для запуску локального серверу використовувалась команда npm start, яка активує Express-

сервер на порту 3001. Після запуску сервер отримує запити від клієнта, обробляє їх, взаємодіє з базою даних і повертає відповідь у форматі JSON.

Завдяки такому середовищу розробки вдалося забезпечити гнучкість, модульність, масштабованість та підтримуваність розробленого ПЗ, а також забезпечити швидку ітерацію та налагодження в процесі реалізації інформаційної системи.

3.2 Вибір і обґрунтування засобів розробки

[4][8] Для реалізації інформаційної системи обліку СТО було обрано стек технологій, який забезпечує ефективну розробку, масштабованість, підтримку сучасних стандартів безпеки та зручну інтеграцію між компонентами.

Вибрані засоби:

- Node.js

Середовище виконання JavaScript на сервері, яке дозволяє створювати продуктивні серверні застосунки. Його неблокуюча модель введення/виведення забезпечує обробку великої кількості одночасних запитів без втрати продуктивності.

- Express.js

Фреймворк для Node.js, який спрощує створення RESTful API, маршрутизацію запитів і обробку даних. Забезпечує просту інтеграцію з базами даних і шаблонами.

- MySQL

Реляційна система управління базами даних, яка забезпечує надійне та швидке зберігання структурованих даних. Має зручний інтерфейс через MySQL Workbench для адміністрування та проєктування.

- HTML, CSS, JavaScript (Vanilla)

Стандартні технології для розробки фронтенд-частини. Забезпечують повну контрольованість візуальної частини системи, зручність адаптації під різні екрани та простоту в налаштуванні інтерактивних елементів.

- Visual Studio Code

Редактор коду з підтримкою розширень, автодоповнення, інтегрованого терміналу, Git і плагінів, що значно полегшує розробку.

- Postman

Інструмент для тестування REST API. Дозволяє перевіряти запити до сервера, зручно налагоджувати та діагностувати помилки.

- Google Chrome + DevTools

Браузер з розширеним інструментарієм розробника для перевірки вмісту сторінок, консольних повідомлень і мережевої активності.

Обґрунтування вибору

Обрані засоби є загальновизнаними стандартами веброзробки. Вони мають активну спільноту, документацію, готові бібліотеки й модулі, що дозволяє прискорити розробку та спростити супровід. Завдяки відкритому коду ці інструменти не потребують ліцензійних витрат.

Розділення на фронтенд та бекенд також дозволяє гнучко змінювати окремі компоненти, не порушуючи роботу всієї системи. Крім того, система може бути легко розгорнута на локальному або хмарному сервері.

3.3 Реалізація клієнтської частини

[5][3] Клієнтська частина інформаційної системи обліку СТО реалізована як набір вебсторінок із використанням HTML5, CSS3 та JavaScript (без фреймворків). Такий підхід забезпечує максимальний контроль над усіма елементами інтерфейсу та дозволяє гнучко реалізувати необхідний функціонал без зайвого перевантаження проєкту.

Основні сторінки інтерфейсу:

- login.html

Сторінка авторизації адміністратора. Реалізовано перевірку введених логіна та пароля, після чого здійснюється перенаправлення до головної сторінки. Для зберігання сесії використовується localStorage.

- index.html

Сторінка керування клієнтами. Містить форму для додавання нового клієнта, список наявних клієнтів, а також кнопки для редагування та видалення. Дані завантажуються з API /api/clients.

- add_vehicle.html

Сторінка додавання автомобілів. Реалізовано динамічне завантаження списку клієнтів, можливість введення марки, моделі та номерного знаку авто. Дані передаються до /api/vehicles.

- services.html

Вивід переліку доступних послуг. Після вибору послуги користувач бачить її ціну та тривалість, які динамічно завантажуються. Дані беруться з API /api/services.

- orders.html

Форма створення нового замовлення. Реалізовано вибір клієнта → авто → послуги (з підтримкою мультивибору). Після надсилання POST-запиту система обчислює загальну вартість замовлення та зберігає його.

- order_list.html

Сторінка перегляду замовлень. Дані підтягуються з сервера і виводяться у вигляді таблиці з інформацією про клієнта, авто, список послуг, дату створення, статус і загальну вартість. Реалізовано кнопку видалення кожного запису.

Додаткові елементи:

- Реалізовано єдину навігацію між сторінками.
- Для зручності користувача використовуються кольорові повідомлення про успішні/помилкові операції.
- Оформлення стилізовано в єдиній кольоровій палітрі (синьо-біло-сірій), що підтримує візуальну цілісність системи.

3.4 Реалізація серверної частини

[7][8]Серверна частина інформаційної системи обліку СТО реалізована за допомогою Node.js з використанням фреймворку Express.js. Вона відповідає за обробку HTTP-запитів від клієнтської частини, взаємодію з базою даних MySQL і повернення відповідей у форматі JSON.

Основні компоненти серверної частини:

- Маршрути (routes):
  - /api/clients — обробка створення, читання, редагування, видалення клієнтів.
  - /api/vehicles — маршрути для додавання авто та фільтрації за клієнтом.
  - /api/services — отримання списку послуг.
  - /api/orders — створення замовлень, вивід усіх замовлень, видалення замовлення.
  - /api/login — маршрут авторизації адміністратора.
- Контролери (controllers):

В кожному маршруті обробка винесена в окремі контролери, де реалізована логіка:

- перевірка коректності вхідних даних;
- взаємодія з базою через ORM Sequelize;
- формування відповідей;
- обробка помилок.

- Моделі (models):

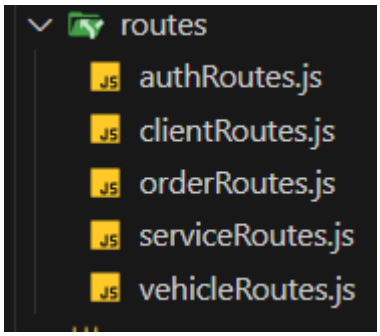
[9] Описані з використанням ORM Sequelize. Усі сутності (Client, Vehicle, Order, Service, OrderService) мають відповідні асоціації:

- Один клієнт має багато авто;
- Одне авто має багато замовлень;
- Одне замовлення містить багато послуг через зв'язуючу таблицю.

◆ База даних:

Підключення реалізовано через файл config/db.js, з параметрами зчитування з .env. Усі операції взаємодії із БД здійснюються через Sequelize-методи (findAll, create, destroy, update, belongsTo, hasMany, belongsToMany).

Структура :



Додаток Б (скріни файлів)

3.5 Тестування та результати

Після завершення етапу розробки клієнтської та серверної частин було проведено тестування інформаційної системи обліку СТО з метою перевірки її працездатності, коректної взаємодії компонентів, а також виявлення можливих помилок і недоліків у логіці виконання функцій.

Мета тестування:

- Перевірити відповідність реалізованого функціоналу вимогам до системи;
- Виявити й усунути помилки у роботі API та інтерфейсу;
- Перевірити стабільність обробки запитів до бази даних;
- Оцінити зручність використання системи кінцевим користувачем.

Проведені види тестування:

- Функціональне тестування
  - Перевірено всі основні маршрути API (/clients, /vehicles, /orders, /services);
  - Перевірено додавання, видалення та редагування клієнтів, авто, замовлень;
  - Перевірено підрахунок загальної вартості послуг у замовленні.
- Тестування форм та валідації
  - Перевірка правильності вводу даних у формах;

- Валідація на стороні клієнта (обов’язковість полів);
- Повідомлення про помилки або успішні дії.

◆ Інтеграційне тестування

- Перевірка взаємодії клієнтської частини з сервером;
- Перевірка відображення динамічних списків клієнтів, авто, послуг;
- Перевірка зв’язків у базі даних після створення замовлення.

◆ Авторизація

- Перевірено вхід до системи з коректними та некоректними даними;
- Працездатність перенаправлення між сторінками залежно від статусу входу.

Результати:

Всі функціональні компоненти системи працюють коректно. У процесі тестування не було виявлено критичних помилок. Система стабільно обробляє запити, правильно оновлює й виводить дані, забезпечуючи зручний інтерфейс для користувача.

Сформовані замовлення правильно зв’язуються з клієнтом, авто і послугами. Дані оновлюються без перезавантаження сторінки, що значно покращує досвід користувача.

Приклад створення Замовлення (Додаток Г)

3.6 Висновки до розділу

У цьому розділі було розглянуто технологічні аспекти реалізації інформаційної системи обліку СТО. Визначено оптимальні інструменти розробки, які були використані для реалізації проєкту, та обґрунтовано їх вибір.

Розглянуто реалізацію клієнтської частини на основі HTML, CSS і JavaScript, яка забезпечує зручний інтерфейс для взаємодії з користувачем. Описано структуру серверної частини, побудованої на платформі Node.js з використанням Express.js та ORM Sequelize для взаємодії з реляційною базою MySQL.

Проведене тестування підтвердило коректну роботу всіх компонентів системи: введення, обробки, збереження та виведення даних. Реалізовані функціональні можливості повністю відповідають вимогам, визначеним на етапі проєктування, а інтерфейс забезпечує комфортну взаємодію з користувачем.

Таким чином, реалізовано технологічно повну систему, готову до використання та подальшого розвитку.

4 ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ТА ЗАХИСТУ ДАНИХ

[2] Захист даних є невід’ємною частиною розробки будь-якої інформаційної системи, що працює з персональною інформацією користувачів та клієнтів. Система обліку СТО зберігає чутливі дані, зокрема імена, контакти клієнтів, номерні знаки авто, а також історію замовлень. Тому впровадження базових принципів безпеки є критично важливим.

4.1 Основні аспекти безпеки

Контроль доступу.

Уся система побудована таким чином, що доступ до функціональності можливий лише після успішної авторизації. Увесь функціонал приховано за механізмом перевірки логіна й пароля адміністратора. Це дозволяє мінімізувати ризик несанкціонованого доступу до чутливої інформації або критичних операцій, таких як видалення чи редагування записів.

Обмеження прав.

Усі CRUD-операції (створення, редагування, видалення) доступні виключно для користувача з правами адміністратора. Інші користувачі, навіть якщо потраплять на сторінку, не зможуть здійснювати жодних дій без відповідної авторизації. Це забезпечує принцип найменших привілеїв, згідно з яким користувачі мають лише той мінімум доступу, який необхідний для виконання їхніх завдань.

Перевірка вхідних даних.

На серверному рівні реалізовано сувору валідацію вхідних запитів. Усі API перевіряють:

- наявність обов’язкових полів (наприклад, client_id, vehicle_id, services);

- коректність формату даних (наприклад, тип поля email — рядок, а не число);

- запобігання ін’єкціям за допомогою валідації строкових та числових полів.

Таким чином, система стійка до типових атак, зокрема SQL-ін’єкцій та XSS-атак.

4.2 Захист API та серверної частини

- CORS (Cross-Origin Resource Sharing).
  Для захисту від запитів, що надходять із небажаних джерел, було налаштовано CORS-політику. Це дозволяє чітко вказати, з яких доменів система може приймати запити. Таке обмеження особливо важливе в контексті захисту від CSRF-атак (Cross-Site Request Forgery).
- Валідація даних.
  Всі дані, що надходять до серверної частини, проходять перевірку на допустимі значення, формати, та довжину. Це запобігає виконанню небезпечних запитів та помилкам у логіці програми.
- ORM Sequelize.
  Замість написання сирих SQL-запитів використовується ORM (Object-Relational Mapping) Sequelize. Це дозволяє уникнути помилок у SQL та значно знижує ризики SQL-ін’єкцій. ORM автоматично екранує значення полів, які вставляються в запити.

4.3 Безпечне зберігання даних

- База даних MySQL.
  Доступ до бази даних забезпечується лише зі сторони сервера, а сам сервер авторизується за допомогою захищеного логіна та пароля, які зберігаються у .env-файлі, що не потрапляє в публічні репозиторії.
- Резервне копіювання.
  Для запобігання втраті важливої інформації реалізовано механізм резервного копіювання. Резервні копії бази даних створюються регулярно й можуть бути збережені як локально, так і на віддалених серверах або у хмарному сховищі. Це дозволяє відновити працездатність системи навіть у випадку збоїв або втрати доступу до основної БД.

4.4 Захист автентифікації

- Локальне збереження авторизації.
  Після успішного входу адміністратора у браузері зберігається прапорець isLoggedIn=true у localStorage, що дозволяє підтримувати сесію користувача між сторінками. Це дозволяє реалізувати "стан входу", зберігаючи зручність для користувача.
- Перевірка доступу на клієнті.
  Перед завантаженням будь-якої функціональної сторінки скрипт перевіряє значення isLoggedIn. Якщо значення відсутнє або має неправильне значення — користувача автоматично перенаправляє на сторінку входу (login.html), чим забезпечується додатковий рівень захисту.

4.5 Можливості подальшого посилення безпеки

Незважаючи на реалізовані заходи, система розроблялася з урахуванням потенційного масштабування, тому передбачено впровадження додаткових засобів захисту:

- Хешування паролів.
  На поточному етапі пароль зберігається у відкритому вигляді. Планується впровадження хешування паролів із використанням бібліотеки bcrypt, що дозволить захистити облікові дані навіть у випадку витоку бази даних.

- Перехід на HTTPS.
  При розгортанні на реальному сервері передбачається використання SSL-сертифіката, що забезпечить шифрування переданих даних, а отже — захист від перехоплення запитів (наприклад, у відкритих Wi-Fi мережах).

- Використання JWT або серверних сесій.
  Для покращення механізму автентифікації передбачено впровадження JSON Web Token (JWT) або сесій на стороні сервера. Це дозволить створювати більш надійний механізм автентифікації, який не залежить від локального сховища браузера.

- Журналювання дій (логування).
  Запровадження системи логування дозволить зберігати інформацію про всі ключові дії в системі: вхід, створення, зміну, видалення даних тощо. Це стане корисним для виявлення підозрілої активності та подальшого аудиту безпеки.

Таким чином, система вже на етапі розробки враховує критично важливі аспекти безпеки, а також має чітку стратегію подальшого зміцнення захисту

даних, що дозволяє адаптувати її для використання в умовах реального підприємства.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було розроблено та реалізовано інформаційну систему обліку Станції Технічного Обслуговування (СТО), яка забезпечує ефективне управління даними про клієнтів, транспортні засоби, послуги та замовлення.

У ході роботи було здійснено:

- аналіз предметної області та виявлення функціональних вимог до системи;
- проєктування структури бази даних, включно з побудовою ER-діаграми;
- розробку архітектури системи за моделлю клієнт-сервер;
- реалізацію клієнтської частини на основі HTML, CSS та JavaScript;
- створення серверної частини на Node.js з використанням Express.js і Sequelize ORM;
- тестування функціоналу та перевірку взаємодії компонентів системи;
- впровадження базових засобів інформаційної безпеки для захисту даних користувачів.

Результатом роботи стала повністю функціональна вебсистема, яка дозволяє адміністратору керувати клієнтами, автомобілями, створювати замовлення з послугами, переглядати історію обслуговування. Система є масштабованою, модульною та придатною для розгортання у реальних умовах.

Отримані результати свідчать про досягнення поставленої мети, відповідність реалізованого програмного забезпечення технічним вимогам і готовність до використання в якості внутрішнього продукту для автоматизації обліку на СТО.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання. – [Чинний від 2016-07-01]. – К.: ДП «УкрНДНЦ», 2016. – 16 с.

2. ISO/IEC 27001:2013. Information technology – Security techniques – Information security management systems – Requirements. – International Organization for Standardization, Geneva, 2013.

3. Флінт, О.В. Веб-програмування: навч. посіб. / О.В. Флінт. – Київ: Видавничий дім «Кондор», 2017. – 320 с.

4. Смірнов, С.В. Основи роботи з MySQL. – Харків: ФОП Бровкін О.В., 2020. – 272 с.

5. Mozilla Developer Network. HTML, CSS, JS documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/

6. Express.js — офіційна документація [Електронний ресурс]. – Режим доступу: https://expressjs.com/

7. Node.js documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/

8. Sequelize – сучасний ORM для Node.js [Електронний ресурс]. – Режим доступу: https://sequelize.org/

9. Кучма, І.В. Проєктування та розробка інформаційних систем: навч. посіб. – Львів: Видавництво Львівської політехніки, 2021. – 284 с.

Додатки

Додаток А

```
const express = require('express');
const router = express.Router();
const authController = require('../controllers/authController');

router.post('/login', authController.login);

module.exports = router;
```

.файл authRoutes.js

```
const express = require('express');
const router = express.Router();
const clientController = require('../controllers/clientController');

console.log('➡️ /api/clients маршрут зареєстровано');

router.get('/', clientController.getAllClients);
router.post('/', clientController.createClient);

router.put('/:id', clientController.updateClient);
router.delete('/:id', clientController.deleteClient);

router.delete('/:id', clientController.deleteClient);

module.exports = router;
```

файл clientRoutes.js

```
const express = require('express');
const router = express.Router();
const orderController = require('../controllers/orderController');

router.post('/', orderController.createOrder);
router.get('/', orderController.getOrders);
router.delete('/:id', orderController.deleteOrder); // 👈 додай цю стрічку

module.exports = router;
```

файл clientRoutes.js

```
const express = require('express');
const router = express.Router();
const serviceController = require('../controllers/serviceController');
```

```
// Отримати всі послуги
router.get('/', serviceController.getAllServices);

// Додати нову послугу
router.post('/', serviceController.createService);

module.exports = router;
```

файл serviceRoutes.js

```
const express = require('express');
const router = express.Router();
const vehicleController = require('../controllers/vehicleController');

console.log('/api/vehicles маршрут зареєстровано');
router.get('/', vehicleController.getVehicles); // 👈 правильно
router.post('/', vehicleController.createVehicle);


module.exports = router;
```

.файл vehicleRoutes.js

Додаток Б

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Авторизація</title>
  <link rel="stylesheet" href="login.css">
</head>
<body>
  <div class="login-container">
    <h2>Вхід до системи</h2>
    <form id="loginForm">
      <label for="username">Логін:</label>
      <input type="text" id="username" required><br>

      <label for="password">Пароль:</label>
      <input type="password" id="password" required><br>

      <button type="submit">Увійти</button>
    </form>
    <p id="error" style="color: red;"></p>
  </div>
```

```
  <script src="login.js"></script>
</body>
</html>
```

файл index.html

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Додати авто</title>
  <link rel="stylesheet" href="add_vehicle.css">
</head>
<body>
   <div class="container">
      <script>
         if (localStorage.getItem('isLoggedIn') !== 'true') {
         window.location.href = 'login.html';
         }
      </script>
      <!-- Перемикач теми
      <div id="themeToggle" class="theme-toggle">
         <img id="themeIcon" src="/frontend/img/sun.png" alt="Theme icon">
      </div> -->
      <nav>
         <a href="index.html">Клієнти</a>
         <a href="add_vehicle.html">Автомобілі</a>
         <a href="services.html">Послуги</a>
         <a href="orders.html">Замовлення</a>
         <a href="order_list.html">Список замовлень</a>
      </nav>

      <h2>Додати нове авто</h2>
      <form id="vehicleForm">
         <input type="text" id="brand" placeholder="Марка" required>
         <input type="text" id="model" placeholder="Модель" required>
         <input type="text" id="plate" placeholder="Номерний знак" required>

         <select id="clientSelect" required>
         <option value="">Оберіть клієнта</option>
         </select>

         <button type="submit">Додати авто</button>
         <hr>
```

```
        <h3>Клієнти та їхні автомобілі</h3>
        <div id="clientsWithVehicles"></div>
      </form>
    <div id="vehicleResult"></div>
  </div>
  <script src="vehicle.js"></script>
</body>
</html>
```

файл add_vehicle.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Список замовлень</title>
  <link rel="stylesheet" href="order_list.css">
</head>
<body>
  <nav>
    <a href="index.html">Клієнти</a>
    <a href="add_vehicle.html">Автомобілі</a>
    <a href="services.html">Послуги</a>
    <a href="orders.html">Замовлення</a>
    <a href="order_list.html">Список замовлень</a>
  </nav>

  <h2>Список замовлень</h2>

  <table id="ordersTable" border="1">
    <thead>
      <tr>
        <th>Клієнт</th>
        <th>Автомобіль</th>
        <th>Послуги</th>
        <th>Дата</th>
        <th>Сума</th>
        <th>Дії</th>
      </tr>
    </thead>
    <tbody>
      <!-- Дані замовлень завантажуються динамічно -->
    </tbody>
  </table>
```

```
  <script src="orders_list.js"></script>
</body>
</html>
```

файл order_list.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Створити замовлення</title>
  <link rel="stylesheet" href="order.css">
</head>
<body>
   <div class="container">
      <nav>
         <a href="index.html">Клієнти</a>
         <a href="add_vehicle.html">Автомобілі</a>
         <a href="services.html">Послуги</a>
         <a href="orders.html">Замовлення</a>
         <a href="order_list.html">Список замовлень</a>
      </nav>
      <script>
   if (localStorage.getItem('isLoggedIn') !== 'true') {
      window.location.href = 'login.html';
   }
      </script>
      <h2>Створити нове замовлення</h2>

      <form id="orderForm">
         <label for="clientSelect">Клієнт:</label>
         <select id="clientSelect" name="client_id" required>
            <option value="">Оберіть клієнта</option>
         </select>

         <label for="vehicleSelect">Автомобіль:</label>
         <select id="vehicleSelect" name="vehicle_id" required>
            <option value="">Оберіть авто</option>
         </select>

         <label for="serviceSelect">Послуги:</label>
         <select id="serviceSelect" name="services" multiple required>
         </select>
```

```
        <button type="submit">Створити замовлення</button>
      </form>

      <p id="orderResult"></p>
         <script>
         if (localStorage.getItem('isLoggedIn') !== 'true') {
            window.location.href = 'login.html';
         }
         </script>
      <script src="orders.js"></script>
   </div>
</body>
</html>
```

файл orders.html