

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ CRM-СИСТЕМИ ДЛЯ
СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ
SOFTWARE IMPLEMENTATION OF A CRM SYSTEM FOR
A SERVICE STATION

Виконав: студент 4 курсу, групи КНТ-221

Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

СУР М.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ГОФМАН Є.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

СУРА Максима Віталійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація CRM-системи для станції технічного обслуговування. Software Implementation of a CRM System for a Service Station

керівник проєкту (роботи) к.т.н., ГОФМАН Євгеній Олександрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “28” квітня 2025 року № 209
2. Строк подання студентом проєкту (роботи) 02 червня 2025 р.

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір програмного забезпечення та його обґрунтування. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

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

7. Дата видачі завдання “ 14 ” квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент

(підпис) Максим СУР
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Євгеній ГОФМАН
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
101 с., 5 табл., 68 рис., 3 дод., 17 джерел.

HTTP, HTTPS, TYPESCRIPT, REACT, REACT TOOLKIT, MONGODB, NODEJS, EXPRESSJS, INTERFACE, SCRIPT, JAVASCRIPT, MERN, JWT, AXIOS, BCRYPT.

Об'єкт дослідження – процес автоматизації управління заявками та клієнтами станції технічного обслуговування (СТО) за допомогою CRM-системи.

Предмет дослідження – методи та технології розробки CRM-системи для СТО, що включають роботу з базою даних клієнтів, управління заявками та послугами. Для реалізації було використано сучасний стек технологій: React, React Toolkit, Express.js, MongoDB, JWT tokens, bcrypt, axios, react-router-dom.

Мета дослідження – створення програмного забезпечення, яке допоможе автоматизувати запис клієнтів, управління послугами та статусами заявок. Це дозволить покращити роботу СТО, зробить процеси швидшими та зручнішими як для менеджерів, так і для клієнтів.

Матеріали, методи та технічні засоби: використання клієнт-серверної архітектури, реалізація авторизації та аутентифікації користувачів. Мова програмування JavaScript, React та Redux Toolkit для фронтенду, серверний фреймворк Express.js, база даних MongoDB, механізми безпеки (JWT, bcrypt), бібліотека axios для взаємодії між клієнтом і сервером. Операційна система Windows 10 та браузері Firefox, Chrome для запуску та тестування застосунку.

Результати. Розроблена CRM-система, яка дозволяє: реєструвати та авторизовувати користувачів, створювати сутності авто-сервісів, менеджерам керувати клієнтами(додавати, редагувати дані), працювати із заявками(створювати, змінювати статус, додавати послуги), вести список

послуг та статусів заявок, гарантувати безпеку даних за допомогою токенів авторизації.

Висновки. У результаті було створено ефективну CRM-систему для станцій технічного обслуговування, яка спрощує управління заявками та клієнтами. Це сприяє підвищенню продуктивності сервісу та полегшує роботу персоналу.

Галузь використання – сервіси технічного обслуговування, які мають проблеми із обліком клієнтів та заявок.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 101 pages, 5 tables, 68 figures, 3 appendixes, 17 sources.

HTTP, ANGULAR, SERVER, NEST, FRONTEND, BACKEND, NODE.JS, TYPESCRIPT, JAVASCRIPT.

The object of research is the process of automating the management of applications and customers of a service station using a CRM system.

The subject of the study is methods and technologies for developing a CRM system for service stations, including working with a customer database, managing requests and services. A modern technology stack was used for implementation: React, React Toolkit, Express.js, MongoDB, JWT tokens, bcrypt, axios, react-router-dom.

The aim of the research is to create software that will help automate customer registration, service management, and application statuses. This will improve the work of service stations, make processes faster and more convenient for both managers and customers.

Materials, methods and tools: use of client-server architecture, implementation of user authorisation and authentication. JavaScript programming language, React and Redux Toolkit for the frontend, Express.js server framework, MongoDB database, security mechanisms (JWT, bcrypt), axios library for client-server interaction. Windows 10 operating system and Firefox and Chrome browsers for launching and testing the application.

Results. A CRM system has been developed that allows you to: register and authorise users, create auto-service entities, manage clients (add, edit data), work with requests (create, change status, add services), maintain a list of services and statuses of requests, and ensure data security using authorisation tokens.

Conclusions. As a result, an effective CRM system for service stations has

been created that simplifies the management of requests and customers. This helps to increase the productivity of the service and facilitates the work of the staff.

The field of application is maintenance services that have problems with customer and request accounting.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	11
Вступ.....	12
1 Аналіз предметної області	13
1.1 Огляд предметної області.....	13
1.2 Актуальність проблеми.....	13
1.3 Огляд аналогів	14
1.3.1 AutoRepair Cloud	14
1.3.2 Auto Repair Bill.....	15
1.3.3 RemOnline	16
1.3.4 Загальне порівняння аналогів	17
1.4 Висновок за розділом 1.....	18
2 Вибір програмного забезпечення та його обґрунтування.....	20
2.1 Аналіз фронтенд фреймворків.....	20
2.1.1 React.....	20
2.1.2 Angular.....	21
2.1.3 Vue.js	21
2.1.4 Загальне порівняння фреймворків	22
2.1.5 Обґрунтування вибору фреймворка для фронтенду	23
2.2 Аналіз бекенд фреймворків.....	24
2.2.1 ExpressJS	24
2.2.2 Django.....	24
2.2.3 Загальне порівняння фреймворків	25
2.2.4 Обґрунтування вибору фреймворка для бекенду.....	26
2.3 Аналіз баз даних для розробки	27
2.3.1 MongoDB.....	27
2.3.2 PostgreSQL	28
2.3.3 SQLite	28
2.3.4 Порівняння баз даних.....	29

2.3.5 Обґрунтування вибору MongoDB.....	30
3 Основні рішення щодо реалізації компонентів системи	32
3.1 Проектування моделей системи	32
3.1.1 Проектування моделей системи на фронтенді.....	32
3.1.2 Проектування моделей системи на бекенді	34
3.1.3 Функціонал реєстрації та авторизації	35
3.1.4 Функціонал додавання запису	37
3.1.5 Пагінація	39
3.2 Проектування графічного інтерфейсу.....	40
3.2.1 Загальна архітектура інтерфейсу.....	40
3.2.2 Основні екрани системи.....	40
3.3 Проектування бази даних	45
3.3.1 Схема User (Користувач).....	45
3.3.2 Схема AutoService (Автосервіс)	45
3.3.3 Схема Client (Клієнт)	46
3.3.4 Схема Record (Заявка/Запис)	46
3.3.5 Options (Налаштування)	47
3.3.6 Загальна структура зберігання даних	47
3.4 Висновки за розділом 3	48
4 Експлуатація, тестування та експериментальне дослідження програми	49
4.1 Призначення й умови застосування програми.....	49
4.2 Вимоги до інтерфейсу.....	50
4.3 Умови виконання програми	51
4.4 Методика та результати тестування.....	52
4.4.1 Google Chrome	53
4.4.2 Mozilla Firefox	60
4.4.3 Microsoft Edge.....	66
4.4.4 Результати тестування	72
4.5 Висновки за отриманими результатами	72
Висновки.....	73

Перелік джерел посилання..... 75

Додаток А Технічне завдання 77

Додаток Б Текст програми 82

Додаток В Слайди презентації..... 93

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;
CRM – Customer Relationship Management;
CRUD – Create, Read, Update, Delete;
CSS – Cascading Style Sheets;
DB – DataBase;
GUI – Graphical User Interface;
HTML – HyperText Markup Language;
JS – JavaScript;
JSON – JavaScript Object Notation;
JWT – JSON Web Token;
REST – Representational State Transfer;
RTK – Redux Toolkit;
SPA – Single Page Application;
TS – TypeScript;
UI – User Interface;
UX – User Experience;
VPS – Virtual Private Server.

ВСТУП

Сучасні інформаційні технології відкривають нові можливості для оптимізації бізнес-процесів, зокрема в сфері станцій технічного обслуговування (СТО). Впровадження CRM-систем допомагає автоматизувати запис клієнтів, управління послугами та обробку заявок, що позитивно впливає на якість сервісу та ефективність роботи автосервісів.

В умовах зростаючої конкуренції автосервіси потребують зручних і гнучких інструментів для ведення бази клієнтів, контролю за виконанням робіт і оцінки якості обслуговування.

У цій дипломній роботі розглядається процес розробки CRM-системи для СТО, яка дозволяє менеджерам створювати та керувати заявками, працювати зі списком послуг, контролювати їх виконання та взаємодіяти з клієнтами. Система реалізована із застосуванням технологій React, React Toolkit, Express.js, MongoDB та інших сучасних інструментів веб-розробки.

Актуальність дослідження пов'язана з потребою автоматизації роботи СТО для підвищення продуктивності, зменшення часу на обробку заявок та покращення комунікації з клієнтами. Запропоноване рішення допомагає ефективно керувати бізнес-процесами автосервісу, оптимізувати роботу персоналу та підвищити рівень задоволеності клієнтів.

Метою роботи є розробка CRM-системи для СТО, яка забезпечує автоматизацію ключових бізнес-процесів, зручний доступ до управління заявками та клієнтською базою, а також безпечну взаємодію між користувачами.

Для досягнення цієї мети у роботі розглядаються сучасні методи та технології веб-розробки, аналізуються особливості CRM-систем у сфері технічного обслуговування автомобілів та створюється відповідне програмне рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд предметної області

СТО є важливими елементами для забезпечення надійності та безпеки автомобілів. Основними функціями СТО є діагностика, ремонт і технічне обслуговування транспортних засобів. З ростом автомобільної індустрії та збільшенням кількості автомобілів, обсяг робіт, які виконують СТО, також зростає. Це спричиняє необхідність оптимізації процесів управління обліком, планування робіт, ресурсами та комунікації з клієнтами.

У сучасному середовищі успішна діяльність СТО неможлива без використання автоматизованих програмних рішень. Тому важливим напрямом розвитку є цифровізація та створення спеціалізованих Customer Relationship Management (CRM) систем, які допомагають ефективно управляти взаємодією з клієнтами, відслідковувати історію звернень, призначати майстрів, контролювати строки виконання робіт і формувати необхідну звітність.

1.2 Актуальність проблеми

Автоматизація бізнес-процесів у сфері технічного обслуговування автомобілів відіграє ключову роль у розвитку сучасних автосервісів. Відсутність єдиної системи для управління клієнтською базою, заявками та послугами може спричинити неефективне використання ресурсів, затримки в роботі та можливі помилки під час взаємодії з клієнтами. Багато станцій технічного обслуговування (СТО) досі покладаються на застарілі методи ведення записів, такі як паперові журнали або електронні таблиці, що ускладнює управління процесами та оперативне реагування на запити клієнтів.

Впровадження CRM-системи для СТО допомагає автоматизувати ці процеси, покращити якість обслуговування та забезпечити швидкий доступ до інформації про заявки й послуги. Інтеграція сучасних веб-технологій дозволяє значно підвищити продуктивність автосервісів та їхню

конкурентоспроможність.

1.3 Огляд аналогів

На ринку існує багато CRM-систем для автосервісів. Нижче наведено три популярних рішення, які широко використовуються для автоматизації процесів технічного обслуговування автомобілів. Оскільки кожне з цих рішень має свої унікальні характеристики, важливо детально проаналізувати їхні переваги та недоліки.

1.3.1 AutoRepair Cloud

AutoRepair Cloud – хмарна CRM-система для автосервісів, яка дозволяє здійснювати управління клієнтами, створення заявок та облік запчастин. Вона орієнтована на невеликі і середні СТО, що шукають простоту у використанні.

Перевагами є:

- хмарне зберігання, а саме, можливість доступу до даних з будь-якого пристрою через Інтернет;
- наявність мобільного додатку дозволяє менеджерам працювати без прив'язки до місця;
- система проста у використанні та не потребує спеціальних навичок;
- можливість приймати онлайн-платежі.

Недоліки:

- у порівнянні з більш потужними системами, AutoRepair Cloud не має таких функцій, як облік складу або детальна аналітика;
- система не дозволяє суттєво адаптуватися до індивідуальних потреб конкретного СТО [1].

Інтерфейс AutoRepair Cloud зображено на рис. 1.1 [2].

Repair Orders #11246445515

2007 Lincoln Town Car 10... More

Add a repair description, ...

Description	Price, \$	Hours / Qty	Total, \$	Status
General Inspection - Full	2.00	2.00		More Save Item
Enter Labor Title - OR - Click Estimate →	Estimate 100	1		More + Add Item
Total		2.00	4.48	

Initial Inspect Approval Get Parts Wait Tech Labor Finished Paid 0.00

Prime Technician: John Dev

Time Started: 2020-06-17 16:05

Requests for Repair / Concerns

- Maintenance every 6 months
- Maintenance every 1 month

Leave Your notes for this repair...

06/17/20 16:13 Mechanic
Hello! The inspection is complete now, some concerns have been added. We are waiting for your approval.

Type your message here...

Help Invoice Workflow Settings Delete Chat

Рисунок 1.1 – Інтерфейс AutoRepair Cloud [2]

1.3.2 Auto Repair Bill

Auto Repair Bill – це хмарне програмне забезпечення для управління автосервісом, яке допомагає механікам та власникам СТО ефективно керувати виставленням рахунків, записами клієнтів, створенням кошторисів, обліком часу та виконаних робіт тощо. Інструмент також дає змогу приймати оплату банківськими картками. Посилання у рахунку дозволяє клієнту оплатити послугу та одразу записатись на наступне обслуговування [3].

До основних переваг системи можна віднести наявність розширених можливостей для фінансового обліку, зокрема ведення витрат та формування рахунків. Ще однією важливою перевагою є те, що десктопна версія програми може працювати без доступу до Інтернету, що особливо актуально для СТО, розташованих у регіонах із нестабільним з'єднанням. Також система підтримує інтеграцію з бухгалтерським обліком, що дозволяє автоматизувати відповідні процеси та підвищити ефективність роботи.

Інтерфейс Auto Repair Bill представлено на рис. 1.2 [4].

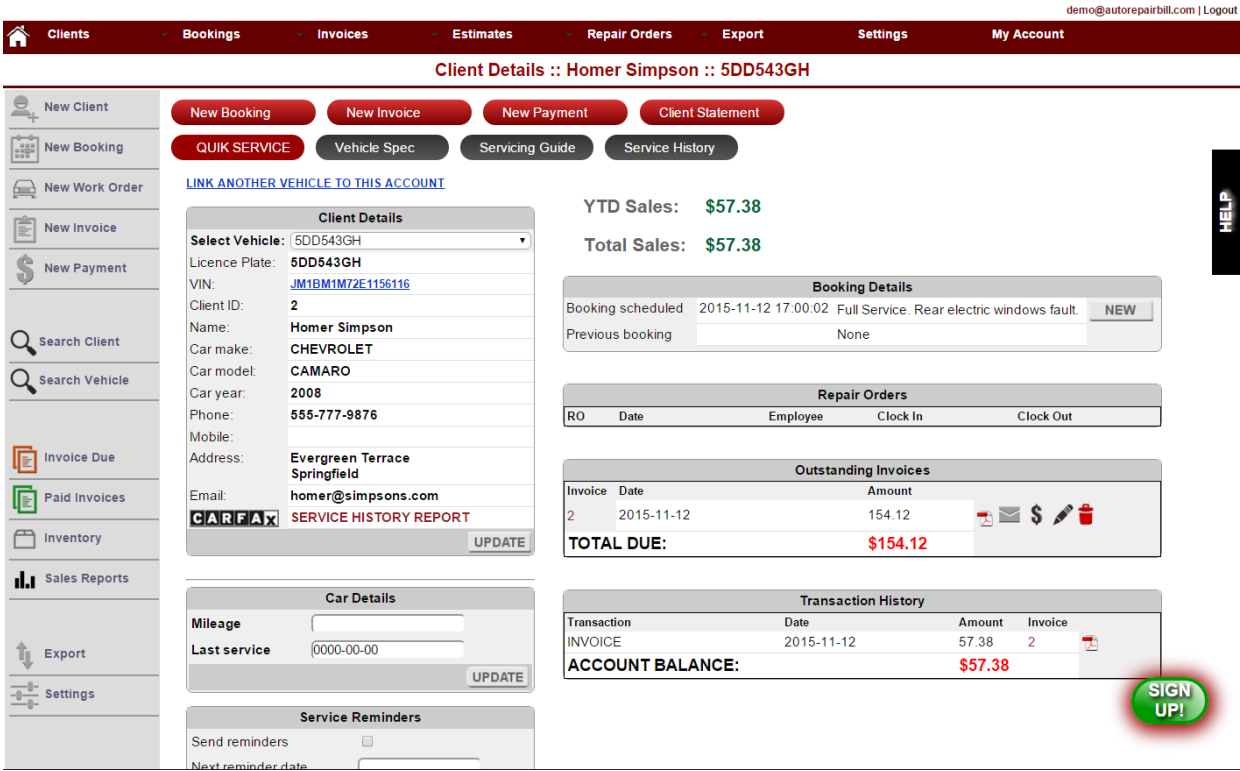


Рисунок 1.2 – Інтерфейс Auto Repair Bill [4]

Серед недоліків варто відзначити високу вартість ліцензії, яка може бути непосильною для невеликих автосервісів. Крім того, відсутність хмарного доступу унеможливорює роботу з системою на відстані, що обмежує мобільність користувачів. Ще одним недоліком є обмежені можливості кастомізації, через що складно налаштувати функціонал під специфічні потреби конкретного підприємства.

1.3.3 RemOnline

RemOnline — це сучасна онлайн-система, яка допомагає автоматизувати управлінські та фінансові процеси в автосервісах. Вона надає інструменти для роботи з клієнтською базою, управління записами, обліку складу, фінансів і аналітичних даних.

Інтерфейс системи відображено на рис. 1.3 [5].

Замовлен...	Строк замове...	Статус	Вибір	Несправність	Клієнт	Виконаве...	Тип замовлення
A2763	01 берез. 2022 14...	Відмова	Vestfrost бн бн3947	не працює хх, мк працює впродовж	Людмила 0000000000	Дмитро Клочко	Не гарантійний, виїзд
A2762	11 д. 01 берез. 2022 14...	В роботі	БН БН бн Т	розмерзся	Тишкевич Ілона 0000000000		Не гарантійний, виїзд
A2761	11 д. 01 берез. 2022 14...	Новий	Kaiser бн бн8053	перестав холодить клацає	Леонид 0000000000		Не гарантійний, виїзд
A2760	11 д. 01 берез. 2022 14...	Очікує погодження	Gorenje бн бн8889	не морозить, світ горить	Рита 0000000000	Дмитро Клочко	Не гарантійний, виїзд
A2759	01 берез. 2022 14...	Доставка	Moulinex 04-S2 503132	втекає через верх	Сабаш Роман 0000000000		Не гарантійний, в се...
A2758	11 д. 01 берез. 2022 14...	В роботі	Samsung WF-E5... Y4T95ABCC0017...	посторонній звук	Козарчук Леонид 0000000000	Дмитро Клочко	Не гарантійний, в се...
A2757	11 д. 01 берез. 2022 14...	Не додзвонились	Grunhelm GWS... GXF804MB901U...	шатається барабан	Будпостач 0000000000		Не гарантійний, в се...
A2756	11 д. 01 берез. 2022 09...	Новий	Panasonic TX-32... 4220167	не вмикається	Банюк Юлія Василів... 0000000000		Не гарантійний, в се...
A2755	11 д. 01 берез. 2022 09...	Очікує запчастину	ATLANT MXM-26... 116271841	не вмикається	Громадський Мико... 0000000000	Дмитро Клочко	Не гарантійний, в се...
A2456/1	10 д. 28 лют. 2022 15:43	В роботі	LG GA-B499YLUZ... 810RFSX11782	не працює	ЛЕЩЕНКО АНТОНИ... 0000000000		Повторний

Рисунок 1.3 – Інтерфейс RemOnline [5]

RemOnline вирізняється широким функціоналом, що охоплює управління складом, фінансами, аналітичними даними, а також підтримує віддалений доступ до системи. Платформа дозволяє легко інтегрувати онлайн-платежі, що спрощує процеси розрахунків із клієнтами. Крім того, вона має гнучкі налаштування, які дають змогу адаптувати систему до специфіки роботи конкретного автосервісу, незалежно від його формату чи масштабу.

Водночас система має і свої недоліки. Одним із них є висока вартість підписки, що може стати бар'єром для невеликих СТО. Ще одним мінусом є те, що процес впровадження та навчання персоналу потребує часу, що може бути проблемою для компаній з обмеженими ресурсами. Також для малих автосервісів частина функціоналу може виявитися надмірною та неактуальною на початкових етапах розвитку бізнесу.

1.3.4 Загальне порівняння аналогів

Для більш детального розгляду та порівняння функціональних можливостей вибраних CRM-систем, наведемо їх характеристики у вигляді

(табл. 1.1).

Таблиця 1.1 – Порівняння аналогів

Критерій	AutoRepair Cloud	Auto Repair Bill	RemOnline
Основні функції	Управління клієнтами, записи на обслуговування, облік запчастин	Виставлення рахунків, контроль витрат, ведення клієнтської бази	Облік клієнтів, управління записами, фінансовий облік, аналітика
Вартість	\$34.99 на місяць	\$200 на одне робоче місце	від €29 на місяць
Гнучкість налаштувань	Середня	Низька	Висока
Інтеграція	Платіжні системи	Бухгалтерський облік	Онлайн-оплати, сторонні сервіси
Тип доступу	Хмарний	Десктопний	Хмарний
Легкість впровадження	Висока	Середня	Висока
Кастомізація	Низька	Низька	Середня

1.4 Висновок за розділом 1

Проведений аналіз предметної області показав, що наявні CRM-системи для автосервісів мають як сильні сторони, так і певні обмеження. Вибір конкретного рішення багато в чому залежить від масштабу бізнесу, специфіки діяльності та фінансових можливостей. Так, AutoRepair Cloud краще підійде для невеликих СТО, де важливими є простота у використанні та доступ з будь-якого пристрою. Car Repair Software більше орієнтована на середні й великі

сервіси, які потребують розширеного фінансового обліку. RemOnline відзначається багатофункціональністю, що робить її ефективним рішенням для великих автосервісів, однак висока вартість підписки може стати перешкодою для малого бізнесу.

Розроблена в межах цієї дипломної роботи власна CRM-система об'єднує переваги розглянутих платформ, зберігаючи при цьому гнучкість і можливість налаштування під конкретні потреби користувачів.

2 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЙОГО ОБҐРУНТУВАННЯ

2.1 Аналіз фронтенд фреймворків

Із зростанням складності сучасних веб-додатків вибір відповідного фронтенд-фреймворку набуває ключового значення. Від цього залежить не лише продуктивність і масштабованість проєкту, а й швидкість розробки та його майбутній розвиток.

Станом на 2024 рік веб-розробка дедалі більше орієнтується на створення динамічних, високопродуктивних та масштабованих інтерфейсів. Сучасні фреймворки повинні забезпечувати підтримку роботи в реальному часі, ефективно обробляти великі обсяги даних та надавати розробникам інструменти, що спрощують створення інтерфейсів, роблячи процес розробки більш зручним та ефективним [6].

2.1.1 React

React (або «ReactJS») — це бібліотека, яка суттєво полегшує створення та управління елементами DOM у веб додатках.

Переваги React:

- бібліотека має невелику вагу — лише близько 2.5 КБ у стисненому форматі Gzip;
- завдяки віртуальному DOM, React забезпечує високу швидкість оновлення інтерфейсу та оптимізований рендеринг. підтримка TypeScript – Angular використовує TypeScript, що забезпечує статичний аналіз коду та допомагає уникнути багатьох помилок ще на етапі розробки;
- синтаксис бібліотеки є інтуїтивно зрозумілим, особливо для тих, хто вже має досвід з JavaScript і HTML. Завдяки цьому поріг входу в роботу з React надзвичайно низький;

– React підтримує TypeScript, зокрема через Create React App та інші CLI-інструменти [7].

2.1.2 Angular

Angular — це фреймворк для створення односторінкових веб-додатків, розроблений на основі TypeScript. Його підтримкою та розвитком займається компанія Google.

Основними перевагами є:

– вбудована підтримка асинхронного програмування за допомогою RxJS. Angular спирається на RxJS — потужну бібліотеку для роботи з асинхронними потоками даних. Це дає змогу додатку реагувати на дії користувача, серверні події та інші зміни без заморожування інтерфейсу;

– розвинений CLI (Angular CLI) як частина екосистеми – Angular має потужний CLI, який дозволяє створювати, налаштовувати і розгортати застосунки за лічені хвилини. CLI автоматизує безліч завдань, включно з оптимізацією продуктивності;

– платформна незалежність (Platform-agnostic philosophy) – Angular можна використовувати разом з NativeScript або Ionic для створення кросплатформних мобільних застосунків. Це дозволяє розробляти рішення для Android і iOS, використовуючи єдину кодову базу;

– підхід до створення односторінкових додатків (SPA) – Angular від самого початку був орієнтований на розробку динамічних SPA, і містить вбудовані засоби для маршрутизації, управління станом, формами, HTTP-запитами, що не потребують сторонніх бібліотек [8].

2.1.3 Vue.js

Vue.js — це поступовий фреймворк, призначений для створення користувацьких інтерфейсів та односторінкових вебзастосунків за допомогою

JavaScript.

До переваг відносяться:

- однофайлові компоненти – підтримка однофайлових компонентів .vue файлів, які об'єднують HTML-шаблон, JavaScript-логіку та стилі в одному місці;

- максимальна гнучкість і поступове впровадження – Vue створено як прогресивний фреймворк: його можна використовувати як окрему бібліотеку для інтеграції в існуючий проєкт або ж як повноцінний фреймворк для розробки з нуля;

- компонентно-орієнтований CSS – Vue дозволяє ізолювати стилі всередині кожного компонента, завдяки чому стилізація одного елементу не впливає на інші частини інтерфейсу [9].

2.1.4 Загальне порівняння фреймворків

В таблиці 2.1 наведено порівняння популярних фреймворків для клієнтської частини проєкту.

Таблиця 2.1 – Порівняння фреймворків для фронтенду

Критерій	React	Angular	Vue
Поріг входу	Середній	Високий	Низький
Підтримка TypeScript	Є, за бажанням	Обов'язкова	Є, але не обов'язкова
Продуктивність	Висока	Висока	Висока
Гнучкість	Висока	Середня	Висока
Розмір збірки	Малий	Великий	Малий
Документація	Дуже добра	Добра, але складна	Чітка і зрозуміла

Продовження таблиці 2.1

Критерій	React	Angular	Vue
Спільнота	Дуже велика	Велика, але менш активна	Середня
Легкість інтеграції	Висока	Середня	Висока

2.1.5 Обґрунтування вибору фреймворка для фронтенду

Після ретельного вивчення різних варіантів фреймворків для розробки свого проєкту, я обрав React завдяки низці переконливих причин, які роблять його ідеальним вибором для моїх потреб:

- гнучкість та адаптивність – React дозволяє створювати компоненти, які можуть бути багаторазово використані в різних частинах проєкту, що дає високу гнучкість і можливість адаптувати його до змін;

- висока продуктивність – віртуальний DOM, який використовує React, дозволяє мінімізувати зміни в реальному DOM, що значно підвищує продуктивність додатка;

- React Native – у разі необхідності створення мобільних застосунків, React дозволяє використовувати React Native для розробки нативних додатків для iOS та Android;

- типізація з Typescript – React має офіційну підтримку TypeScript, що дозволяє використовувати статичну типізацію для підвищення надійності коду та зменшення кількості помилок при розробці.

2.2 Аналіз бекєнд фреймворків

Ландшафт розробки серверної частини переживає радикальні зміни, оскільки ми занурюємося в 2025 рік. Вже минули ті часи, коли вибір фреймворку для серверної частини обмежувався лише найпопулярнішими варіантами. Сьогодні розробники стикаються з більш складним вибором, де потрібно знайти баланс між традиційною стабільністю та сучасними вимогами до продуктивності [10].

2.2.1 ExpressJS

ExpressJS являє собою фреймворк для платформи Node.js, що пропонує широкий набір можливостей і переваг для створення веб-додатків.

Переваги фреймворку:

- мінімалістичність та легкість – Express.js — це легкий фреймворк, який надає лише базовий функціонал для створення веб-застосунків;
- простота у вивченні та використанні – Express має інтуїтивно зрозумілий API, що робить його зручним для розробників різного рівня підготовки;
- підтримка middleware-функцій – однією з ключових особливостей Express є можливість використання middleware — функцій, що обробляють HTTP-запити. Їх можна використовувати для ведення логів, перевірки автентичності, обробки помилок тощо;
- масштабованість – завдяки неблокуючій моделі введення/виведення, Express чудово справляється з великою кількістю одночасних підключень [11].

2.2.2 Django

Django — це повноцінний фреймворк для веб-розробки, орієнтований на використання мови програмування Python. Його головне призначення —

спростити та прискорити процес створення веб-застосунків шляхом надання широкого набору вбудованих інструментів.

Основні переваги використання:

- розширена безпека за замовчуванням – Django забезпечує високий рівень безпеки одразу «з коробки»;
- вбудована система автентифікації та керування користувачами – Django містить потужну й гнучку систему керування користувачами, яка дозволяє створювати, автентифікувати та авторизовувати користувачів без необхідності додаткових бібліотек;
- висока структурованість проєкту – Django пропонує чітко визначену структуру проєкту та дотримується принципів "конвенції понад конфігурацію";
- докладна та стандартизована документація – Django має одну з найкращих систем документації серед open-source фреймворків. Вона охоплює всі аспекти роботи з фреймворком і особливо корисна для початківців [12].

2.2.3 Загальне порівняння фреймворків

Порівняльна характеристика двох популярних вебфреймворків — Django та Express.js — наведена у (табл. 2.2).

Таблиця 2.2 – Порівняльна таблиця фреймворків

Характеристика	ExpressJS	Django
Мова програмування	JavaScript (Node.js)	Python
Архітектура	Мінімалістичний, без чіткої структури	MVC (Model-View-Controller)
Поріг входу	Помірний. Добре підходить тим, хто вже знайомий з JavaScript	Вбудовані, включаючи підтримку WebSockets, мікросервісів та GraphQL

Продовження таблиці 2.2

Характеристика	ExpressJS	Django
Безпека	Потребує ручної реалізації механізмів захисту	Має вбудовані засоби захисту від CSRF, XSS, SQL-ін'єкцій та ін.
Вбудований ORM	Немає. Потрібно використовувати сторонні (Sequelize, Mongoose тощо)	Є потужний вбудований ORM
Масштабованість	Підходить для мікросервісної архітектури, масштабування вручну	Підходить для великих проєктів, але масштабування складніше

2.2.4 Обґрунтування вибору фреймворка для бекенду

Рішення вибору ExpressJS для серверної частини базується на ряді важливих переваг:

- мінімалізм – Express.js є мінімалістичним і модульним фреймворком, що дозволяє створювати серверну частину застосунку з нуля, надаючи максимальну гнучкість для налаштування під специфічні потреби проєкту;

- висока продуктивність – Express підходить для проєктів середнього та малого масштабу, таких як CRM-система для СТО, забезпечуючи високу швидкість обробки запитів при помірному навантаженні;

- гнучкість в архітектурі – Express не накладає жорстких обмежень на структуру застосунку, що дозволяє розробникам самостійно обирати найбільш підходящий підхід для організації коду;

- ідеальний для роботи з JavaScript-стеком – Express.js безперешкодно інтегрується з іншими частинами JavaScript-стеку, такими як React на

фронтенді, що сприяє спрощенню комунікації між різними частинами проєкту.

2.3 Аналіз баз даних для розробки

У світі розробки програмного забезпечення вибір правильної бази даних є критично важливим рішенням, яке може суттєво вплинути на продуктивність, масштабованість і зручність вашого застосунку. Зважаючи на різноманітність доступних варіантів, іноді складно визначити, яка система управління базами даних (СУБД) найкраще відповідатиме вашим вимогам [13].

2.3.1 MongoDB

MongoDB — це документна NoSQL база даних. Це відкритий інструмент, який використовується для зберігання великих обсягів даних на основі документів.

До переваг належать:

- рівень продуктивності – MongoDB зберігає більшу частину даних в оперативній пам'яті (RAM), що забезпечує швидшу роботу під час виконання запитів;
- гнучкість – схема MongoDB не є заздалегідь визначеною, що дозволяє працювати з динамічною архітектурою даних, що зберігаються;
- шардинг – якщо сервер не може впоратися з великим обсягом даних, MongoDB автоматично розподіляє їх, не зупиняючи роботу;
- масштабованість – завдяки шардингу MongoDB використовує горизонтальну масштабованість, на відміну від SQL-баз, що застосовують вертикальну масштабованість;
- простота – MongoDB пропонує простий синтаксис запитів, який значно легший для освоєння, ніж SQL [14].

2.3.2 PostgreSQL

PostgreSQL — це одна з найстаріших, але водночас найсучасніших систем управління базами даних з відкритим кодом. Вона підтримується та обслуговується активною незалежною спільнотою, яка надає допомогу та підтримку для вирішення питань при роботі з PostgreSQL. Цю систему можна запускати на різних платформах, таких як Windows, Linux та Mac.

До переваг належать:

- транзакції — PostgreSQL підтримує транзакційні DDL-операції (створення, видалення таблиць), що дозволяє виконувати зміни в базі даних у межах однієї транзакції, що важливо для складних реляційних застосунків;
- параметри — База даних має велику кількість налаштовуваних параметрів, які можна змінювати на різних рівнях (сесія, транзакція, функція), що дає більшу гнучкість у налаштуванні роботи системи;
- розширюваність — PostgreSQL є високо розширюваною: можна додавати нові функції, типи даних, мови програмування, що дозволяє адаптувати базу даних під специфічні потреби користувача;
- функції безпеки — База даних надає потужні механізми безпеки, включаючи параметричну безпеку та контроль доступу на рівні користувачів. Це дозволяє налаштувати безпеку на рівні бази даних та додатків [15].

2.3.3 SQLite

SQLite є дуже популярною базою даних, яка успішно використовується у вигляді файлів на диску для настільних додатків, таких як системи контролю версій, інструменти для фінансового аналізу, програми для каталогізації та редагування медіа, пакети для САПР, програми для ведення обліку тощо.

До переваг належать:

- легкість та компактність — SQLite є дуже легкою базою даних, що

робить її ідеальною для вбудованого використання в пристроях, таких як телевізори, мобільні телефони, камери, побутова електроніка тощо;

- швидкість – операції читання та запису виконуються дуже швидко, що робить SQLite на 35% швидшою за файлові системи;

- доступність – Базу даних SQLite можна легко відкривати через багато сторонніх інструментів, що робить її зручною для користувачів, які хочуть працювати з даними через зовнішні інтерфейси;

- без необхідності в установці – достатньо завантажити бібліотеки SQLite, і вона готова до створення бази даних без складних налаштувань;

- портативність – SQLite є портативною і працює на всіх 32-бітних та 64-бітних операційних системах, а також на архітектурах з великим і малим порядком байтів [16].

2.3.4 Порівняння баз даних

Таблиця 2.3 порівнює основні характеристики таких популярних баз даних, як MongoDB, PostgreSQL, і SQLite.

Таблиця 2.3 – Порівняння баз даних

Критерій	MongoDB	PostgreSQL	SQLite
Тип бази даних	NoSQL, документна база даних	Реляційна база даних	Легка, файлова реляційна база даних
Продуктивність	Висока продуктивність	Висока продуктивність,	Швидша в 35% за файлові системи

Продовження таблиці 2.3

Критерій	MongoDB	PostgreSQL	SQLite
Масштабованість	Горизонтальна масштабованість через шардінг	Вертикальна масштабованість, підтримка реплікацій	Обмежена масштабованість
Інсталяція	Вимагає налаштування серверів і додаткових компонентів	Потрібно встановити на сервері або локально	Не потребує інсталяції, легко інтегрується
Транзакції	Підтримуються	Обмежена підтримка	Підтримуються
Гнучкість схеми	Без схеми, динамічні документи	Строга схема, підтримка складних типів даних	Простота в схемі, підтримка невеликих змін
Безпека	Підтримка автентифікації, авторизації, шифрування	Розширена безпека на рівні ролей і привілеїв	Параметри безпеки на рівні ОС та додатків

2.3.5 Обґрунтування вибору MongoDB

У порівнянні з іншими базами даних, MongoDB має декілька ключових переваг, які роблять її найкращим вибором для реалізації CRM-системи для СТО:

- це документна база даних, яка дозволяє зберігати дані у форматі JSON-подібних документів;

- дає можливість зберігати дані безпосередньо в пам'яті, що значно прискорює операції читання та запису порівняно з традиційними реляційними базами даних;
- однією з основних переваг MongoDB є її можливість горизонтального масштабування за допомогою шардінгу;
- MongoDB добре інтегрується з популярними технологіями, такими як JavaScript, Node.js та сучасні фреймворки для розробки веб-додатків;
- MongoDB має простий і зрозумілий синтаксис запитів, що робить її легшою для освоєння порівняно з SQL.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Проектування моделей системи

Важливим етапом є розуміння структури і взаємодії компонентів на фронтенді та бекенді. Аналізуючи моделі системи на цих двох рівнях, можна краще зрозуміти, як дані передаються між користувачем і сервером, як відбувається їх обробка та як інтеграція різних елементів додатку сприяє створенню ефективної і масштабованої системи. У наступному розділі ми детально розглянемо моделі на фронтенді та бекенді, звертаючи увагу на важливість кожного з компонентів.

3.1.1 Проектування моделей системи на фронтенді

Фронтенд моделі системи — це концептуальна структура, яка описує, як компоненти та стани взаємодіють з користувачем, а також як зберігаються та обробляються дані. Оскільки я використовую кастомні хуки та Redux Toolkit, розглянемо основні етапи цього процесу.

Кастомні хуки

Кастомні хуки — це функції, які дозволяють абстрагувати логіку взаємодії з даними та іншими функціональностями, спрощуючи управління станом і взаємодію з компонентами. Ось основні кастомні хуки, які використовуються в системі:

- `useClients`: хук для отримання списку клієнтів із бекенду та управління їхнім станом. Він виконує API-запит для отримання даних та оновлює локальний стан для відображення останніх змін;

- `useClientsModal`: хук для керування станом модального вікна клієнтів. Він відповідає за відкриття/закриття модальки та управління даними, що редагуються чи додаються;

- `useModal`: загальний хук для керування будь-якими модальними вікнами в системі. Він дозволяє відкривати або закривати модальні вікна за

допомогою внутрішнього стану;

- `useRecordsModal`: хук для роботи з модальним вікном записів клієнтів. Він керує відкриттям і закриттям вікна для перегляду чи редагування записів, а також виконує асинхронні операції для їх оновлення чи видалення;

- `useRecordsService`: хук для обробки даних про записи клієнтів. Він дозволяє створювати, оновлювати та видаляти записи, централізуючи роботу з сервісами та забезпечуючи безпеку взаємодії з бекендом;

- `useRecordStatus`: хук для керування статусами записів (наприклад, від "Новий" до "Виконано"). Він відповідає за оновлення статусів та синхронізацію даних через API;

- `useRouter`: хук для роботи з маршрутизацією в додатку. Дозволяє програмно переміщатися між сторінками або компонентами, що корисно для навігації між записами або модальними вікнами.

Redux Toolkit (RTK)

Redux Toolkit — це офіційний набір інструментів для роботи з Redux, який спрощує налаштування та використання Redux у вашому проекті. Він допомагає мінімізувати кількість необхідного коду, що дозволяє зосередитись на важливій логіці додатку, без необхідності конфігурувати Redux вручну. Це включає в себе автоматичне налаштування сховища, надання зручних інструментів для оновлення стану та підтримку потужних можливостей для роботи з незмінними даними [17].

Основні компоненти RTK у системі:

- `Slice`: кожен `slice` відповідає за певну область стану додатку. Наприклад, існують окремі `slice`-файли для клієнтів (`clientsSlice.tsx`), записів (`recordsSlice.tsx`), авторизації (`authSlice.tsx`) тощо. У кожному `slice` визначаються ред'юсери для обробки дій, таких як створення, редагування або видалення даних, а також реалізуються асинхронні функції (`thunks`) для взаємодії з сервером;

- `Thunks`: використовуються для виконання асинхронної логіки,

наприклад, запитів до API або реалізації складних бізнес-процесів перед оновленням стану Redux. У застосунку `thunks` можуть виконувати такі завдання, як: завантаження списку клієнтів з сервера, створення нових записів для клієнтів, оновлення статусів записів, обробка можливих помилок під час запитів;

- **Selectors**: функції, які дозволяють отримати конкретні частини стану з Redux store. Вони допомагають ізолювати доступ до стану, дозволяючи компонентам отримувати лише потрібні дані без прямої взаємодії з усім глобальним сховищем.

3.1.2 Проєктування моделей системи на бекенді

На серверній частині системи моделі визначають, яким чином організовані та зберігаються дані в базі, а також встановлюють правила взаємодії з цими даними. У вашому випадку бекенд побудований на `Express.js`, а для роботи з `MongoDB` використовується бібліотека `Mongoose`, що дозволяє створювати схеми для опису структури документів.

Ключові моменти бекенду:

- **User**: відповідає за збереження інформації про користувачів системи та реалізацію механізмів авторизації й автентифікації;
- **Client**: модель клієнта автосервісу, яка включає інформацію про власника, його автомобілі та персональні знижки;
- **Record**: визначає записи про надані клієнтам послуги, включаючи типи робіт, їхню вартість та інші деталі обслуговування;
- **AutoService**: представляє сам автосервіс із прив'язкою до клієнтів та переліку доступних послуг;
- **Options**: модель, що містить глобальні налаштування системи — наприклад, статуси обслуговування або список доступних послуг.

API на сервері забезпечує взаємодію з клієнтською частиною додатку

через обробку таких запитів, як:

- отримання списку записів для конкретного клієнта;
- створення нових або редагування вже існуючих записів;
- зміна статусів записів відповідно до етапу обслуговування.

3.1.3 Функціонал реєстрації та авторизації

Функціональність реєстрації та авторизації у CRM-системі для СТО реалізована з акцентом на безпеку, зручність користування та ефективну взаємодію між фронтендом і бекендом. Форми входу та реєстрації представлені у вигляді окремих компонентів `registration.tsx` та `auth.tsx`. Вони містять стандартні поля, такі як електронна пошта, пароль. Перед надсиланням даних на сервер здійснюється базова валідація, а у разі помилок користувач отримує відповідне повідомлення.

На клієнтській стороні використовується централізоване управління станом через `Redux Toolkit`. Вся логіка авторизації зосереджена у `authSlice`, який зберігає інформацію про поточного користувача, статус входу та можливі помилки. Асинхронні санки `loginUser` і `registerUser` (файл `authThunks.tsx`) відповідають за взаємодію з сервером. У разі успішного входу, згенерований JWT-токен і дані користувача зберігаються в `localStorage`, що дозволяє зберегти сесію навіть після оновлення сторінки. Цей токен автоматично додається до HTTP-запитів, що забезпечує авторизований доступ до захищених API.

Під час реєстрації створюється не лише обліковий запис, а й ініціалізується базова структура даних — формується нова сутність `AutoService`, до якої згодом прив'язуються клієнти, записи та налаштування. Після успішної реєстрації користувач автоматично авторизується, що забезпечує плавний перехід до системи без додаткового входу.

На сервері реалізована відповідна логіка обробки запитів. Під час створення користувача його пароль хешується за допомогою `bcrypt`, а у відповідь формується JWT-токен. При вході система перевіряє наявність

користувача у базі, порівнює пароль з хешованим значенням та у разі успішної авторизації також видає токен клієнту.

Після входу користувача перенаправляє до головної сторінки управління (ControlPage), яка містить навігаційне меню для доступу до клієнтів, записів, налаштувань тощо. Якщо ж користувач неавторизований, система автоматично обмежує доступ до захищених сторінок і перенаправляє на форму входу.

Реалізована система реєстрації та авторизації є надійною, масштабованою і гармонійно інтегрована в загальну архітектуру проєкту, забезпечуючи як захист даних, так і зручність для кінцевого користувача. Блок-схеми функціоналу авторизації та реєстрації зображені на рис. 3.1 та рис. 3.2 відповідно.



Рисунок 3.1 – Блок-схема авторизації



Рисунок 3.2 – Блок-схема реєстрації

3.1.4 Функціонал додавання запису

Процес створення нової заявки в CRM-системі для СТО є важливою складовою взаємодії між адміністратором та клієнтськими записами. Ця функціональність включає зручний інтерфейс та потужну логіку обробки даних як на клієнтській, так і на серверній стороні.

Інтерфейс створення заявки реалізовано в модальному вікні, яке відкривається з головної сторінки розділу записів (records). Компонент, відповідальний за відображення цього вікна, знаходиться у файлі `recordsModal.tsx`. Він містить форму для введення інформації, де адміністратор може обрати існуючого клієнта або додати нового, вказати марку та номер автомобіля, вибрати послуги з цінами, зазначити дату та час (від і до), а також додати додатковий опис заявки. Крім того, після створення заявки, адміністратор може вибрати статус заявки з попередньо налаштованого списку.

Дані, введені в форму, обробляються через кастомний хук `useRecordsModal`, який керує локальним станом модального вікна, його відкриттям/закриттям та збереженням введених значень. Обрані послуги динамічно додаються через хук `useRecordsService`, що дозволяє управляти списком послуг та розраховувати загальну вартість. Статуси заявки, доступні для вибору, отримуються через хук `useRecordStatus`, який витягує список статусів з глобального стану, визначених адміністратором у налаштуваннях.

Після заповнення форми, коли адміністратор натискає кнопку збереження, дані заявки передаються до санка `createRecord`, який знаходиться у файлі `recordsThunks.tsx`. Цей санк формує запит до сервера з усіма даними заявки, включаючи ідентифікатор клієнта, дату та часовий проміжок. Дані додаються до відповідного дня (`day`) у схемі `AutoService.days` і прикріплюються до клієнта. У разі успіху заявка додається до глобального стану `Redux` через відповідний ред'юсер у файлі `recordsSlice.tsx`.

На сервері контролер заявки приймає ці дані, перевіряє їх на валідність та додає до бази даних `MongoDB` через модель `AutoService`. Заявка зберігається як вкладений документ в масиві `days.records`, що дозволяє групувати заявки по датах. Якщо клієнта з таким ідентифікатором немає, система автоматично створює нового клієнта та прив'язує його до сервісу.

Інтерфейс системи підтримує інтерактивне оновлення: після створення заявки `UI` автоматично оновлюється, і користувач може побачити нову заявку на сторінці записів, у відповідний день. Для відображення кожної заявки використовується компонент `RecordCard`, який виводить основну інформацію про заявку у вигляді картки з можливістю редагування або перегляду деталей. Блок-схему створення запису показано на рис. 3.3.



Рисунок 3.3 – Блок-схема створення нового запису

3.1.5 Пагінація

У модулі records CRM-системи реалізована базова пагінація для ефективного відображення великої кількості клієнтських записів. Такий механізм дозволяє оптимізувати інтерфейс, зменшити навантаження на браузер і забезпечити швидку та зручну роботу з даними.

На фронтенді пагінація реалізована шляхом розбиття масиву records на окремі сторінки. Компонент, що відповідає за виведення записів, зберігає стан поточної сторінки та кількість записів, які відображаються одночасно. При зміні сторінки оновлюється відповідний фрагмент даних, що потрапляє до рендеру.

Користувач має змогу перемикатися між сторінками за допомогою елементів навігації — стрілок або номерів сторінок — розміщених у нижній частині інтерфейсу. Ці елементи передають індекси для оновлення currentPage у відповідному slice стану.

3.2 Проектування графічного інтерфейсу

Інтерфейс користувача (UI) у CRM-системі для СТО створений з акцентом на інтуїтивність, зручність та орієнтацію на повсякденні бізнес-процеси. Його головна задача — надати швидкий і легкий доступ до клієнтів, записів, переліку послуг та налаштувань, водночас мінімізуючи потребу користувача в технічних знаннях.

3.2.1 Загальна архітектура інтерфейсу

Інтерфейс системи побудований на основі React із використанням TypeScript. Усі компоненти згруповані в папці `components` та організовані за принципом розподілу обов’язків. Контейнерні компоненти відповідають за логіку відображення і керування даними, наприклад, у модулях `records`, `clients` чи `options`. Презентаційні компоненти зосереджені на візуальній частині інтерфейсу та взаємодії з користувачем, як-от `cards`, `modal` або `clientsmodal`. Крім того, у проєкті використовуються утилітарні компоненти на кшталт `LIST` для навігації або `header` для відображення заголовків сторінок, які забезпечують підтримку основного функціоналу. Загальна архітектура компонентів зображена на рис. 3.4.

3.2.2 Основні екрани системи

У `App.tsx` організована базова структура з використанням `BrowserRouter` та умовним рендерингом компонентів залежно від шляху. Роутинг реалізовано просто: шлях веде до `ControlPage`, де відбувається подальша внутрішня навігація. Основні екрани системи зображено на рис. 3.5-3.10.

На рис. 3.4 зображена архітектура компонентів, на рис. 3.5 зображений екран системи.

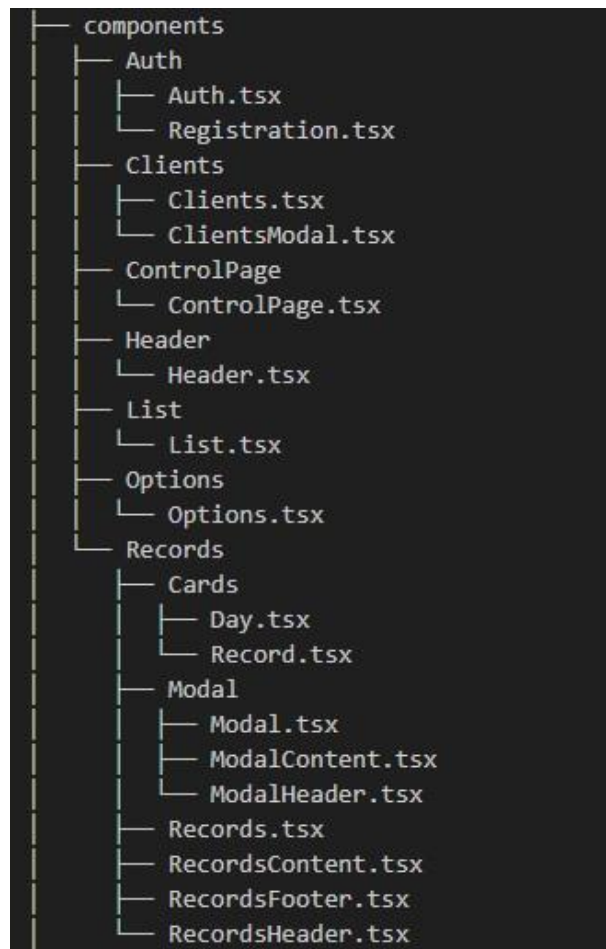


Рисунок 3.4 – Загальна архітектура компонентів

AutoService CRM

Email

Пароль

Увійти Зареєструватися

Рисунок 3.5 – Екран авторизації

На рис. 3.6-3.7 зображено екрани системи.

AutoService CRM

Email

Пароль

Підтвердіть пароль

Зареєструватися

Авторизуватися

Рисунок 3.6 – Екран реєстрації

Запис

Запис

Клієнти

Налаштування

Вийти

Календар

Записати клієнта

20 квітня

12:00-15:00

Чекати записування

Погодити

BMW X7 - Максим

asdsadas

Заміна колодок - 500

Разом: 500

Закрити заявку

< Попередній

Наступний >

Рисунок 3.7 – Екран записів

На рис. 3.8-3.9 зображено екрани системи.

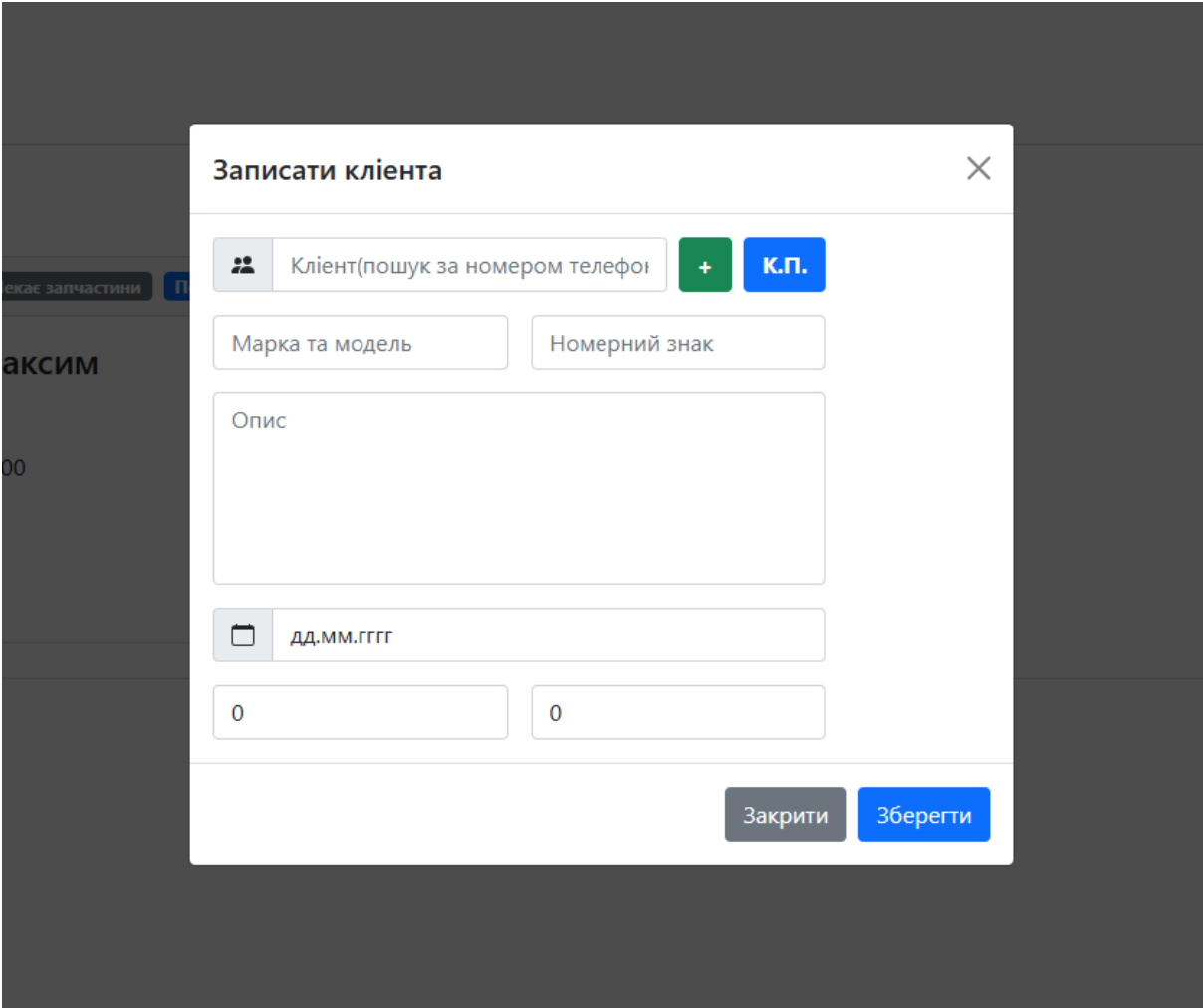


Рисунок 3.8 – Модальне вікно створення запису

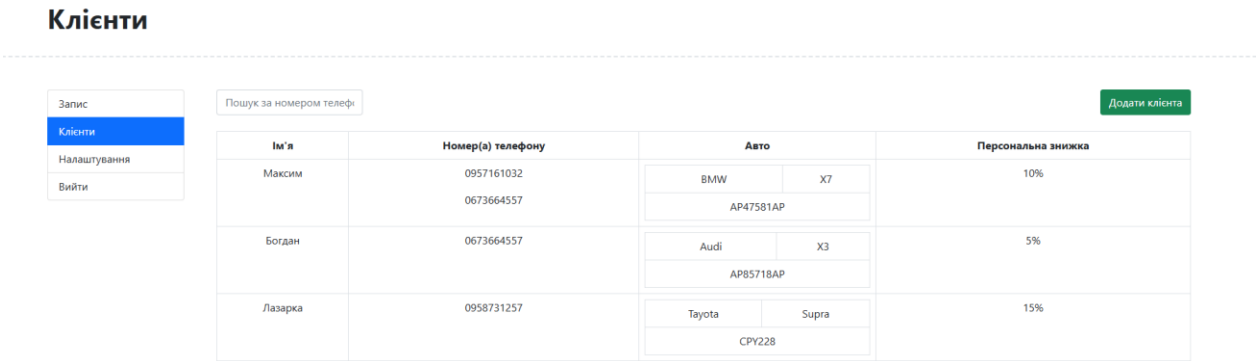


Рисунок 3.9 – Екран клієнтів

На рис. 3.10-3.11 зображено екрани системи.

Створити клієнта

Ім'я

Марка

Модель

Номерний знак

Телефон 1

Телефон 2 (опційно)

0

Закрити

Зберегти

Рисунок 3.10 – Модальне вікно створення клієнта

Налаштування

Запис

Клієнти

Налаштування

Вийти

Додавання нового статусу заявки

Новий статус заявки

Додати

Додавання нової послуги

Послуга

0

Додати

Рисунок 3.11 – Екран налаштувань автосервісу

3.3 Проєктування бази даних

У розробленій CRM-системі для станції технічного обслуговування основним компонентом для зберігання та обробки даних є база даних MongoDB, яка є документно-орієнтованою нереляційною системою. MongoDB була обрана через свою гнучкість у роботі з вкладеними структурами даних, здатність до масштабування та легкість інтеграції з Node.js за допомогою бібліотеки Mongoose.

3.3.1 Схеми User (Користувач)

Ця модель відповідає за зберігання облікових записів працівників чи власників сервісу.

Ключові поля:

- email: зберігає електронну адресу користувача;
- password: зберігає пароль користувача;
- autoServiceId: зберігає ідентифікатор автосервісу до якого належить користувач.

Паролі зберігаються у ґешованому вигляді за допомогою bcrypt, що забезпечує базовий рівень безпеки.

3.3.2 Схеми AutoService (Автосервіс)

Центральна модель, що містить усі записи, клієнтів та налаштування, пов'язані з певною станцією технічного обслуговування.

Ключові поля:

- name: зберігає ім'я автосервісу;
- days: зберігає вкладений масив днів який містить інформацію про робочі дні з вкладеними заявками;
- clients: зберігає вкладений масив який містить інформацію про

клієнтів;

- options: зберігає вкладений масив який містить інформацію про налаштування автосервісу (послуги та статуси заявок).

3.3.3 Схеми Client (Клієнт)

Описує клієнтів, що користуються послугами сервісу.

Ключові поля:

- name: зберігає ім'я клієнта;
- phoneNumber: зберігає масив який зберігає номери мобільного телефону клієнта;
- car: зберігає об'єкт який має поля (brand, model, number) що містять інформацію про транспортний засіб клієнта;
- personalDiscount: зберігає персональну знижку клієнта.

3.3.4 Схеми Record (Заявка/Запис)

Ця модель представляє собою заявку на обслуговування.

Ключові поля:

- clientName: зберігає ім'я замовника(клієнта);
- clientId: зберігає ідентифікатор клієнта який міститься у схемі client;
- car: зберігає назву та модель транспортного засобу клієнта;
- carNumber: зберігає номер транспортного засобу клієнта;
- services: зберігає масив послуг у заявці;
- totalCost: зберігає загальну вартість робіт;
- finalCost: зберігає фінальну вартість робіт;
- description: зберігає опис заявки;
- date: зберігає дату заявки;
- duration: зберігає об'єкт який має інформацію про час виконання

заявки;

- status: зберігає статус заявки.

Ці записи зберігаються вкладено в дні у моделі AutoService.

3.3.5 Options (Налаштування)

Модель зберігає доступні для користувача варіанти статусів заявки (наприклад, “Новий”, “У роботі”) та доступні типи послуг з відповідними цінами

Ключові поля:

- statusWorkOptions: зберігає масив статусів заявки;
- servicesOptions: зберігає масив послуг які в свою чергу мають поля назви послуги та вартості.

3.3.6 Загальна структура зберігання даних

База даних організована таким чином, щоб мінімізувати кількість запитів. Наприклад, замість окремої колекції для записів (records), вони зберігаються безпосередньо в моделі AutoService в межах відповідного дня (days). Таке вкладене зберігання зручне для отримання записів за конкретний день, оскільки дає змогу одразу витягувати весь масив записів.

Архітектура бази даних поєднує класичне документне зберігання з вкладеними структурами, що забезпечує оптимальний баланс між ефективністю доступу до даних та гнучкістю в розширенні функціоналу. Усі моделі взаємопов'язані через autoServiceId, що формує чітку ієрархію та обмежує обсяг доступних даних для кожного користувача, підвищуючи рівень безпеки.

3.4 Висновки за розділом 3

У цьому розділі розглянуто основні технічні рішення, що лягли в основу реалізації CRM-системи для СТО. Проаналізовано як фронтенд-, так і бекенд-структури, що дало змогу зрозуміти принципи зберігання, обробки та передачі даних. Особливу увагу приділено архітектурі взаємодії інтерфейсу з даними за допомогою Redux Toolkit і кастомних хуків, що підвищило масштабованість та спростило підтримку.

Інтерфейс розроблено з урахуванням зручності користувача: використовуються модальні вікна, таблиці, картки та сторінки для керування клієнтами й записами. Структура бази даних побудована на MongoDB, що забезпечує гнучкість, масштабованість і адаптацію до змін бізнес-процесів. Прийняті рішення базуються на сучасних підходах до розробки веб-додатків, що забезпечує надійність, ефективність і зручність у подальшому розвитку системи.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Розроблена CRM-система орієнтована на автоматизацію ключових процесів, які щоденно відбуваються на станції технічного обслуговування (СТО). Її головна мета — забезпечити ефективне ведення обліку клієнтів, контроль за заявками та наданими послугами, а також надати зручний інтерфейс для повсякденної роботи співробітників сервісного центру.

Це веб-застосунок, що функціонує у браузері та доступний із будь-якого пристрою, підключеного до Інтернету. Система побудована за принципами клієнт-серверної архітектури: фронтенд реалізований за допомогою React у поєднанні з Redux Toolkit для управління станом, а бекенд — на Node.js із використанням Express і бази даних MongoDB. Така структура дозволяє гнучко масштабувати проєкт і при необхідності інтегрувати його з іншими сервісами.

Основними користувачами виступають адміністратори або менеджери СТО, які проходять авторизацію через спеціальний інтерфейс та отримують доступ до панелі керування. У межах системи реалізовано можливість додавання нових клієнтів, створення заявок із зазначенням дати, часу, типу послуги та статусу, а також редагування, фільтрації й видалення даних. Навігація між записами здійснюється через пагінацію, що робить роботу з великими обсягами інформації швидкою й зручною.

Для використання системи достатньо стандартного сучасного комп'ютера або ноутбука з доступом до Інтернету. Серверну частину можна розмістити як на локальному обладнанні, так і на віддаленому хостингу типу VPS.

Серед переваг — легке встановлення, простота підтримки, доступний і зрозумілий інтерфейс, а також можливість швидкої адаптації до змін у бізнес-процесах. Упровадження цієї CRM-системи дозволяє оптимізувати обробку клієнтських заявок, зменшити кількість помилок та втрат даних, забезпечити

зручний доступ до історії обслуговування та загалом підвищити ефективність роботи СТО.

4.2 Вимоги до інтерфейсу

Інтерфейс користувача є одним із ключових факторів, що визначають зручність і продуктивність роботи з CRM-системою. Під час його розробки основна увага приділялася простоті, логічності у використанні, швидкому доступу до основного функціоналу та відповідності сучасним принципам UI/UX-дизайну.

Центральним елементом інтерфейсу виступає головна панель керування, яка містить навігаційне меню (реалізоване через компонент List) для переходу між основними розділами, такими як «Клієнти», «Записи» та «Опції». У верхній частині кожної сторінки динамічно відображається заголовок активного розділу за допомогою компонента Header.

У розділі клієнтів реалізовано зручне відображення списку у вигляді таблиці, з можливістю додавання нових записів та редагування вже наявних через модальне вікно ClientsModal. Система автоматично перевіряє унікальність введених даних, забезпечує форматування номерів телефонів і дозволяє здійснювати пошук за іменем або номером авто.

Сторінка з записами побудована на основі карток днів (DayCard), у кожній з яких відображаються відповідні заявки клієнтів. Для кожного запису використовується компонент RecordCard, що містить дані про клієнта, транспортний засіб, обрану послугу та статус. Додавання, редагування або видалення запису здійснюється через модальне вікно RecordsModal, де доступний вибір із заздалегідь заданих опцій.

Для спрощення управління станом інтерфейсу та взаємодії з Redux були створені кастомні React-хуки, зокрема useModal, useClients і useRecordsModal. Вони дозволяють централізовано контролювати логіку відкриття та закриття модальних вікон, а також обробку клієнтських і записних даних.

Інтерфейс побудований на основі компонентного підходу з використанням React і стилізований згідно з єдиною дизайн-системою, що забезпечує послідовність та візуальну цілісність усіх елементів.

4.3 Умови виконання програми

Система функціонує в середовищі сучасного веб-браузера і не вимагає встановлення додаткового програмного забезпечення на комп'ютер користувача, оскільки реалізована як клієнт-серверний веб-застосунок. Умови її роботи поділяються на вимоги до клієнтської (фронтенд) та серверної (бекенд) частин.

З боку користувача достатньо мати пристрій із базовими технічними:

- характеристиками (процесор від 1.5 ГГц, не менше 4 Гб оперативної пам'яті та близько 300 МБ вільного дискового простору);
- стабільне підключення до Інтернету (якщо сервер розміщено віддалено) та встановлений сучасний веб-браузер такий як Google Chrome, Mozilla Firefox або Microsoft Edge.

Для коректного відображення інтерфейсу та взаємодії з користувачем браузер повинен підтримувати сучасні веб-технології, зокрема JavaScript стандарту ES6 і вище, а також CSS Grid і Flexbox. Бажаною є також підтримка localStorage або sessionStorage, що дозволяє тимчасово зберігати дані авторизації, наприклад JWT-токен.

З боку сервера система повинна запускатися у середовищі Node.js (версії 16 і вище). Для повноцінної роботи необхідно мати встановлені Node.js і npm, а також локальну або хмарну базу даних MongoDB (наприклад, через MongoDB Atlas). Серверна частина вимагає налаштування середовища — конфігураційного файлу .env, який містить ключові параметри доступу до бази даних і секрети для роботи з токенами. Запуск сервера виконується стандартними командами на кшталт npm start або node index.js.

Також система може бути розгорнута на віртуальному сервері (VPS), де

для забезпечення безперебійної роботи доцільно використовувати менеджери процесів, наприклад pm2, які автоматизують моніторинг і перезапуск застосунку у разі збоїв.

CRM-система не має високих вимог до ресурсів, тому може стабільно працювати навіть на недорогому обладнанні. Її продуктивність дозволяє ефективно обробляти великий обсяг даних у реальному часі, забезпечуючи комфортну роботу користувачам.

4.4 Методика та результати тестування

Для перевірки коректної роботи графічного інтерфейсу CRM-системи було проведено тестування на основі ручного перегляду функціональності у трьох основних сучасних браузерів:

- Google Chrome (версія 135.0.7049.85);
- Mozilla Firefox (версія 137.0.1);
- Microsoft Edge (версія 135.0.3179.73).

Тестування здійснювалося з метою виявлення можливих помилок відображення та перевірки стабільності взаємодії користувача з інтерфейсом. Я орієнтувався на ключові критерії, що безпосередньо впливають на зручність користування та функціональність:

- коректне відображення інтерфейсу — перевірялося правильне завантаження та візуальне оформлення основних сторінок (реєстрація, авторизація, робота з клієнтами, записами та налаштуваннями);

- збереження токена — після авторизації здійснювалася перевірка наявності JWT-токена в локальному сховищі браузера (localStorage) для забезпечення стабільної сесії користувача;

- взаємодія з модальними вікнами — оцінювалась коректність відкриття, закриття, наповнення полів, а також відправлення даних через модальні форми (створення клієнтів, записів);

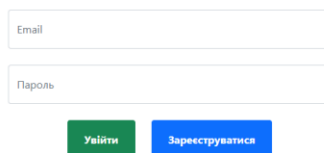
– валідація полів — перевірялась наявність валідації для обов’язкових полів та коректне реагування інтерфейсу на помилки введення (наприклад, некоректний номер телефону або відсутність обов’язкового значення).

4.4.1 Google Chrome

Перевіримо коректне відображення графічного інтерфейсу, валідацію полів, збереження токenu авторизації та взаємодію з модальними вікнами з використанням браузеру Google Chrome. Перевірку коректного відображення графічного інтерфейсу та валідацію полів наведено на рис. 4.1-4.7.

На рис. 4.1 зображено екран системи.

AutoService CRM



Email

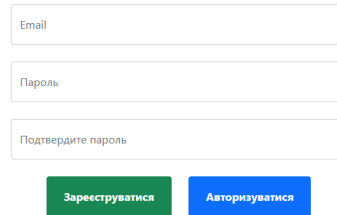
Пароль

Увійти Зареєструватися

Рисунок 4.1 – Сторінка авторизації

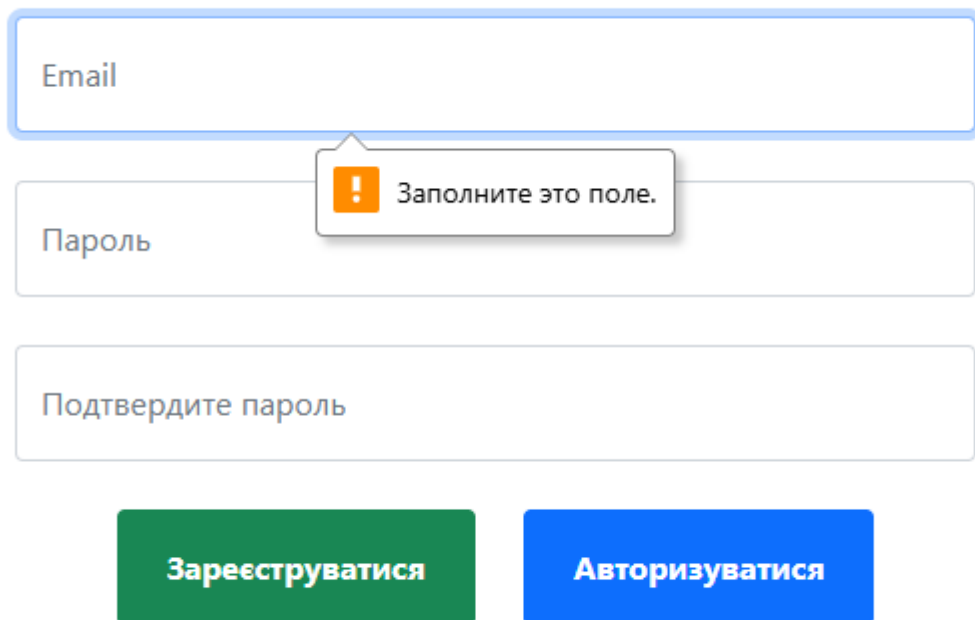
На рис. 4.2-4.3 зображено екран системи та валідацію поля відповідно.

AutoService CRM



Registration form with three input fields: "Email", "Пароль" (Password), and "Подтвердите пароль" (Confirm password). Below the fields are two buttons: "Зареєструватися" (Register) in green and "Авторизуватися" (Log in) in blue.

Рисунок 4.2 – Сторінка реєстрації



Registration form with three input fields: "Email", "Пароль" (Password), and "Подтвердите пароль" (Confirm password). The "Email" field is highlighted with a blue border. A tooltip with an orange exclamation mark icon and the text "Заполните это поле." (Fill in this field.) points to the "Пароль" field. Below the fields are two buttons: "Зареєструватися" (Register) in green and "Авторизуватися" (Log in) in blue.

Рисунок 4.3 – Попередження про незаповнене поле при реєстрації

На рис. 4.4-4.6 зображено валідація полів та екрани системи відповідно.

Email

Пароль

Увійти

Зареєструватися

!

Заполните это поле.

Рисунок 4.4 – Попередження про незаповнене поле при авторизації

Запис

Календар

Записати клієнта

Клієнти

Налаштування

Вийти

20 квітня

12:00-15:00

Час запису

Послуги

BMW X7 - Максим

aridcaas

Завірка колодок - 500

Разом: 500

Зберегти запис

< Попередній

Наступний >

Рисунок 4.5 – Сторінка записів

Клієнти

Пошук за номером телеф

Додати клієнта

Запис

Клієнти

Налаштування

Вийти

Ім'я	Номер(а) телефону	Авто	Персональна знижка
Максим	0957161032 0673664557	BMW X7 AP47581AP	10%
Богдан	0673664557	Audi X3 AP85718AP	5%
Лазарка	0958731257	Tayota Supra CPY228	15%

Рисунок 4.6 – Сторінка клієнтів

На рис. 4.7-4.8 зображено екран системи та процес пошуку токену відповідно.

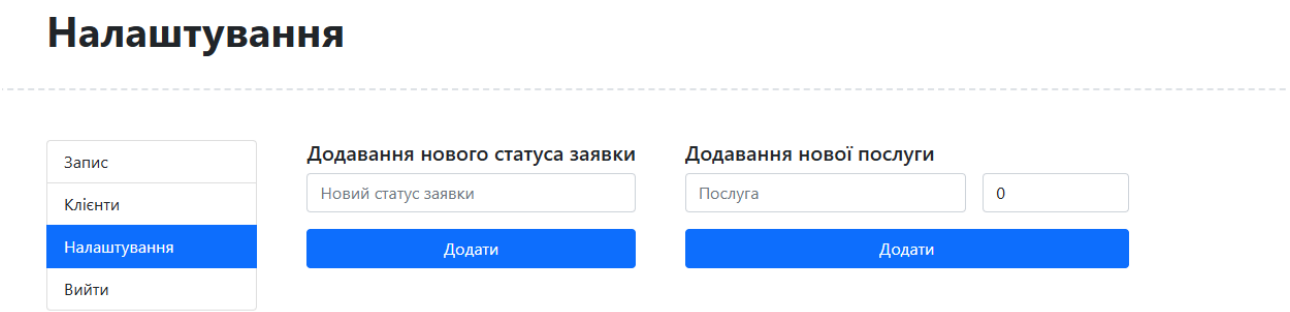


Рисунок 4.7 – Сторінка налаштувань

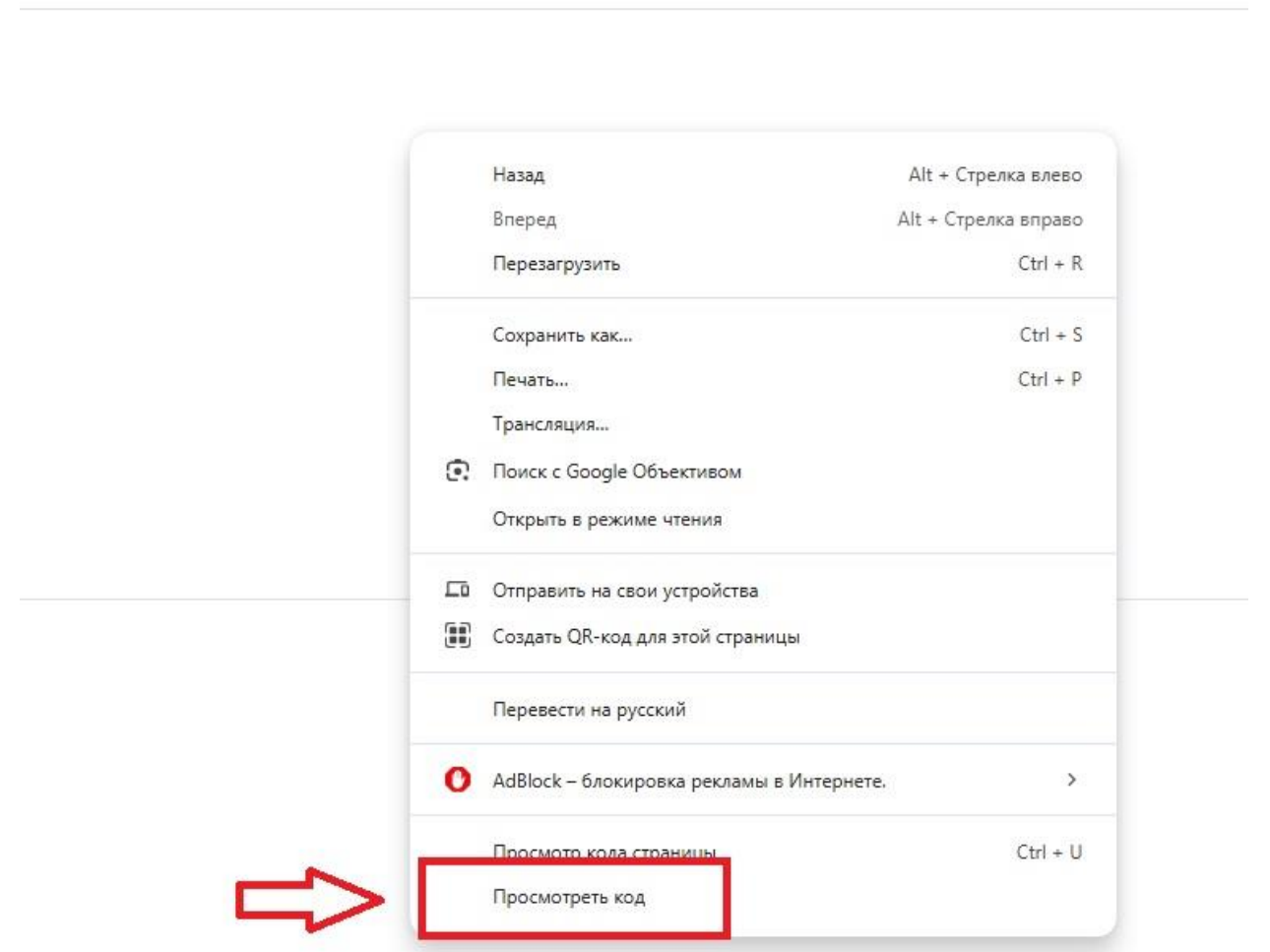


Рисунок 4.8 – Меню

На рис. 4.9-4.10 зображено процес пошуку токену.

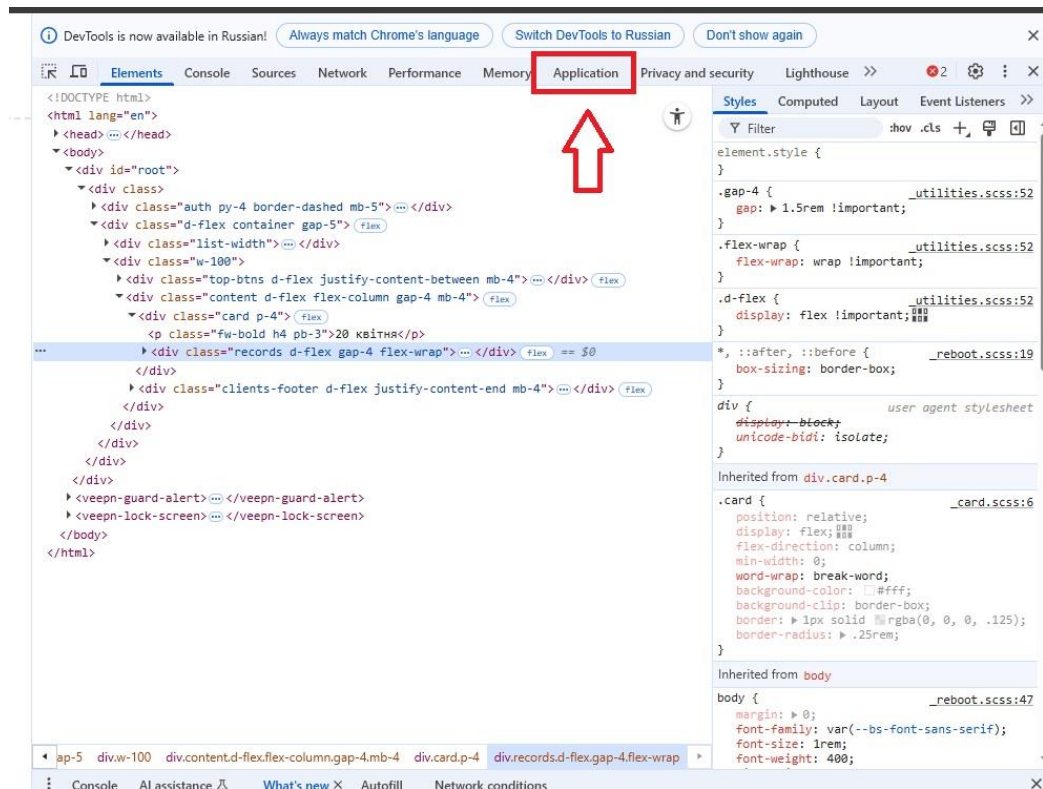


Рисунок 4.9 – Редактор коду

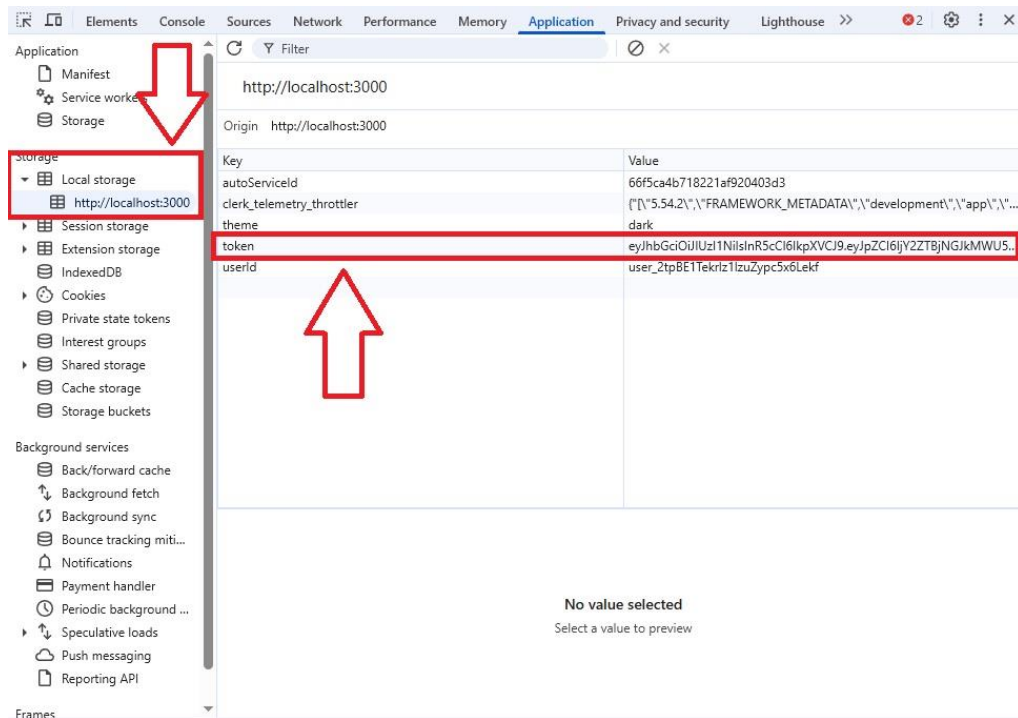
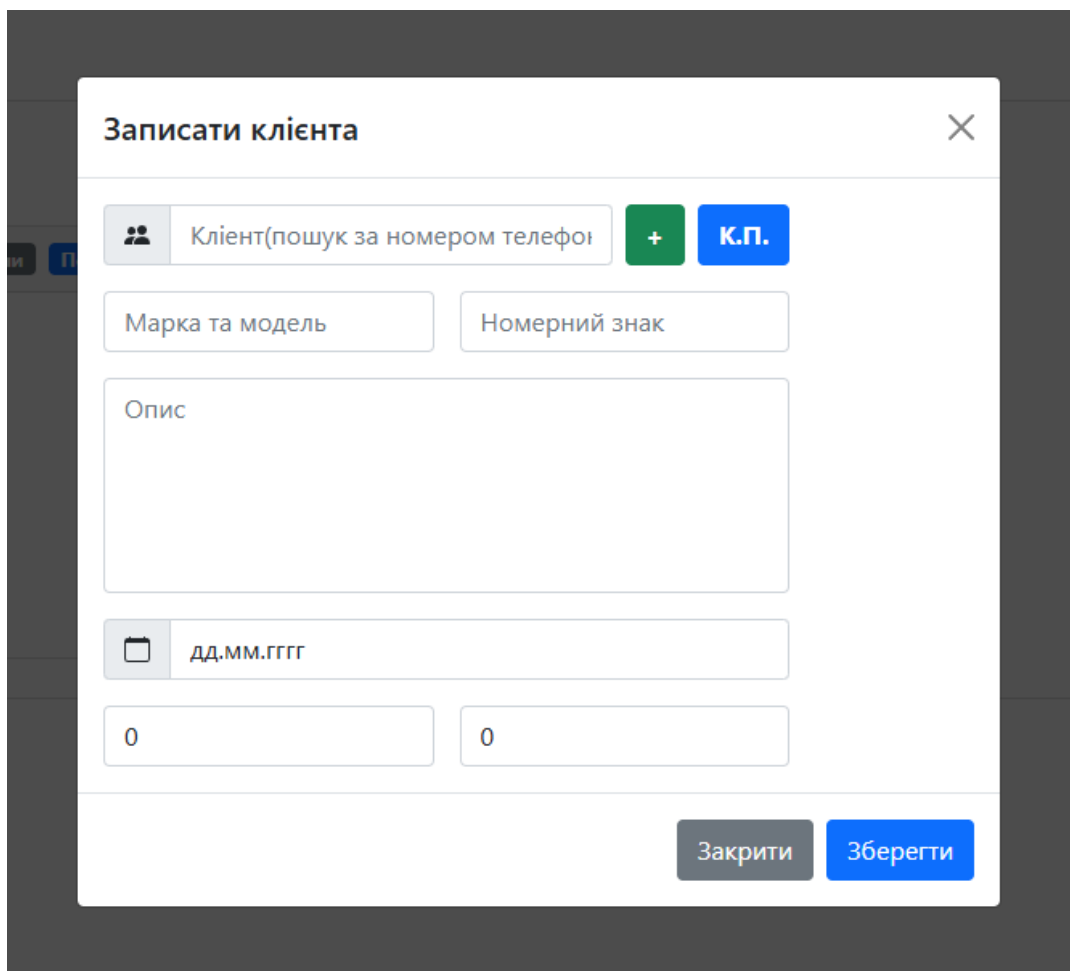


Рисунок 4.10 – Пункт Application

У результаті тестування графічний інтерфейс та функціонал валідації продемонстрували очікувані результати та поведінку. Перейдемо до збереження токена. Для того щоб перевірити чи зберігається токен авторизації у браузері Google Chrome потрібно: натиснути правою кнопкою миші по екрану, обрати пункт «Подивитися код», перейти у пункт меню «Application» та знайти там пункт Local storage. Увесь процес перевірки токена наведено на рис. 4.8-4.10.

При переході на вкладку Local storage ми можемо побачити, що токен коректно зберігається у локальному сховищі браузера. І останнім пунктом, перевіriamo роботу із модальними вікнами. Повну перевірку зображено на рис. 4.11-4.13. На рис. 4.11 зображено екран системи.



Записати клієнта

Клієнт(пошук за номером телефону) + К.П.

Марка та модель

Номерний знак

Опис

ДД.ММ.ГГГГ

0 0

Закрити Зберегти

Рисунок 4.11 – Модальне вікно запису клієнта

На рис. 4.12-4.13 зображено екрани системи.

Записати клієнта

Максим

+

К.П.

BMW X7

AP47581AP

тест

30.04.2025

9

12

Закрити

Зберегти

Рисунок 4.12 – Заповнене модальне вікно запису клієнта

30 квітня

9:00-12:00

Новий

Послуги

BMW X7 - Максим

тест

Закрити заявку

Рисунок 4.13 – Успішно створений запис

У результаті тестування інтерфейс продемонстрував стабільну роботу на всіх вказаних браузерах, а також повну відповідність поставленим функціональним вимогам.

4.4.2 Mozilla Firefox

Перевіримо коректне відображення графічного інтерфейсу, валідацію полів, збереження токена авторизації та взаємодію з модальними вікнами з використанням браузеру Mozilla Firefox. Перевірку коректного відображення графічного інтерфейсу та валідацію полів наведено на рис. 4.14-4.20. На рис. 4.14-4.15 зображено екрани системи.

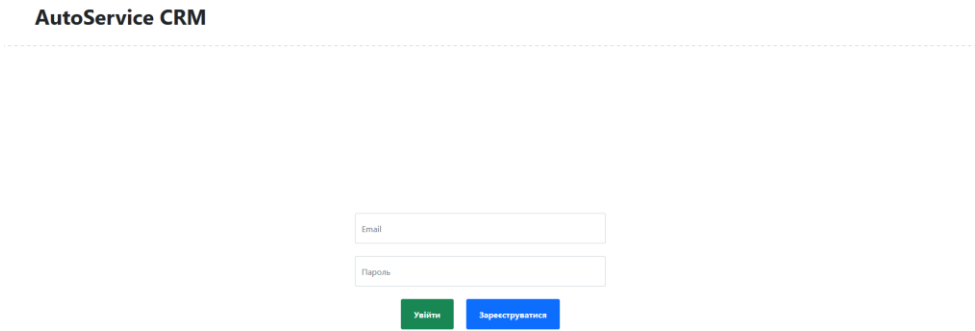


Рисунок 4.14 – Сторінка авторизації

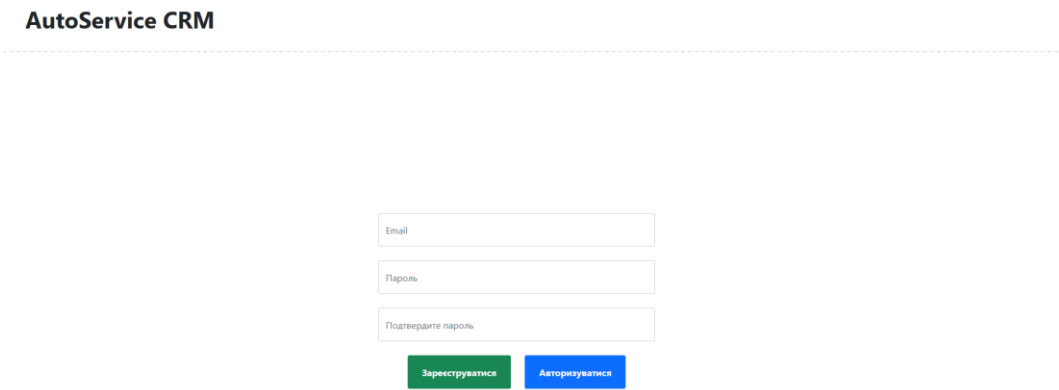


Рисунок 4.15 – Сторінка реєстрації

На рис. 4.16-4.17 зображено валідацію полів.

The image shows a registration form with three input fields. The first field, labeled 'Email', is highlighted with a blue border. A tooltip message 'Пожалуйста, заполните это поле.' (Please fill out this field.) is displayed over the second field, which is labeled 'пароль' (password). The third field is labeled 'Подтвердите пароль' (Confirm password). Below the fields are two buttons: a green button labeled 'Зареєструватися' (Register) and a blue button labeled 'Авторизуватися' (Log in).

Рисунок 4.16 – Попередження про незаповнене поле при реєстрації

The image shows the same registration form as in Figure 4.16. In this version, the 'Email' field is filled with the text 'Email'. The tooltip message 'Пожалуйста, заполните это поле.' (Please fill out this field.) is now displayed over the second field, which is labeled 'пароль' (password). The buttons below are a green button labeled 'Увійти' (Log in) and a blue button labeled 'Зареєструватися' (Register).

Рисунок 4.17 – Попередження про незаповнене поле при авторизації

На рис. 4.18-4.20 зображено екрани системи.

Запис

Запис

Клієнти

Налаштування

Вийти

Календар

Залишити клієнта

20 квітня

12:00-15:00

Час запису

Послуга

BMW X7 - Максим

asdasdas

Залишок кошика - 500

Разом: 500

Закрити заявку

30 квітня

9:00-12:00

Послуга

Послуга

BMW X7 - Максим

тест

Закрити заявку

< Попередній

Наступний >

Рисунок 4.18 – Сторінка записів

Клієнти

Запис

Клієнти

Налаштування

Вийти

Пошук за номером телеф

Додати клієнта

Ім'я	Номер(а) телефону	Авто	Персональна знижка
Максим	0957161032	BMW X7	10%
	0673664557	AP47581AP	
Богдан	0673664557	Audi X3	5%
		AP85718AP	
Лазарка	0958731257	Tayota Supra	15%
		CPV228	

Рисунок 4.19 – Сторінка клієнтів

Налаштування

Запис

Клієнти

Налаштування

Вийти

Додавання нового статусу заявки

Новий статус заявки

Додати

Додавання нової послуги

Послуга

0

Додати

Рисунок 4.20 – Сторінка налаштувань

На рис. 4.21-4.23 изображений процесс пошуку токену.

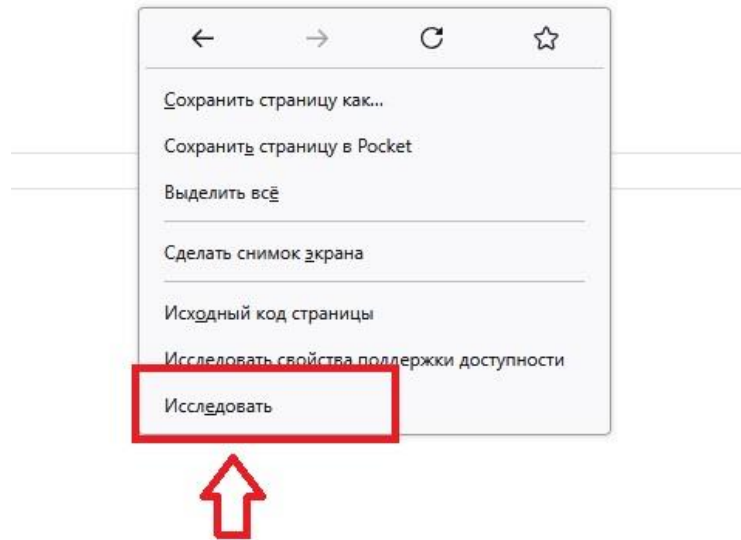


Рисунок 4.21 – Выпадающее меню

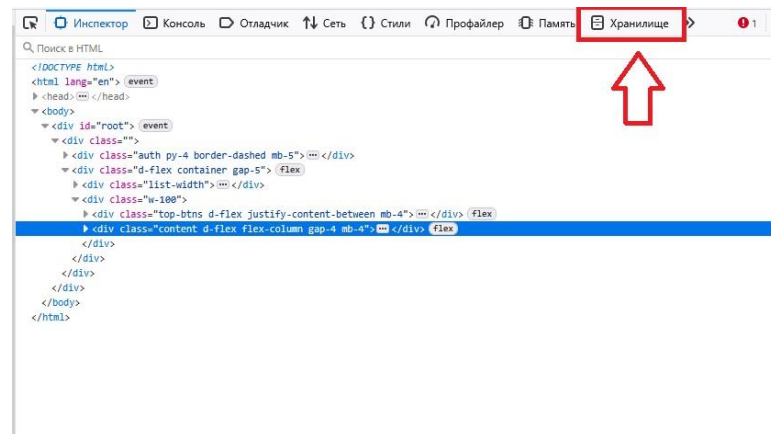


Рисунок 4.22 – Редактор коду

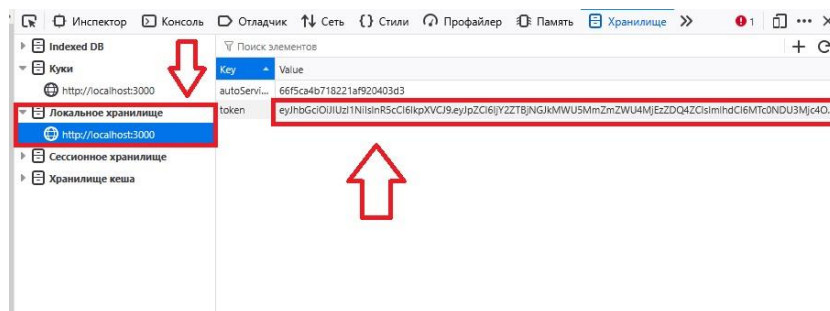
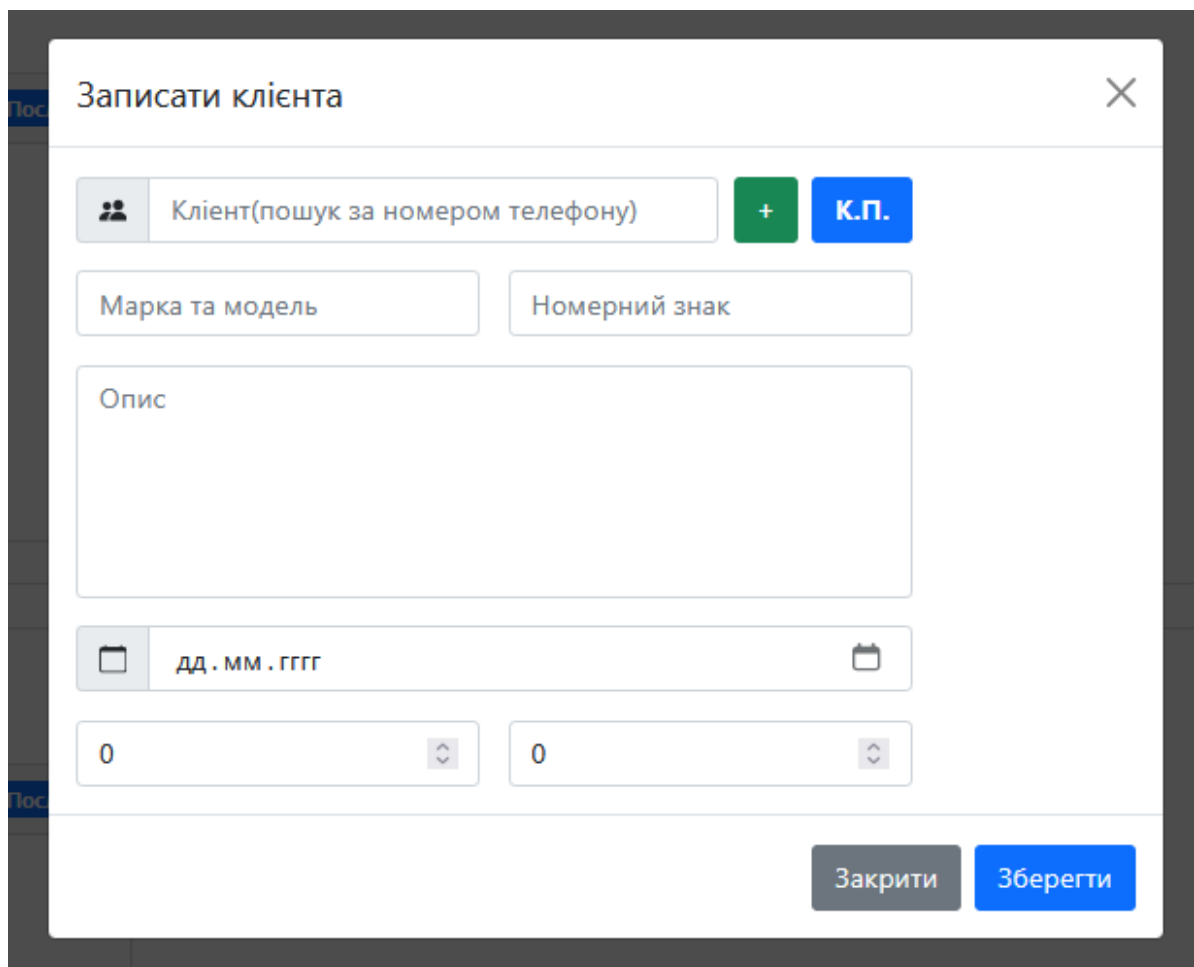


Рисунок 4.23 – Вкладка локального сховища

У результаті тестування були помічені відмінності з інтерфейсом у браузері Google Chrome, а саме: шрифти часу запису були не жирними. В цілому, функціонал авторизації та вигляд інтерфейсу показали задовільні результати. Перейдемо до збереження токена, щоб перевірити чи зберігається токен у браузері Mozilla Firefox потрібно: натиснути праву кнопку миші по екрану, перейти в пункт «Дослідити», потім у вкладку «Сховище» и знайти там «Локальне сховище». Процес перевірки збереження токена зображено на рис. 4.21-4.23.

Останнім пунктом залишилося перевірити правильну роботу модальних вікон. Тестування модальних вікон наведено на рис. 4.24-4.26. На рис. 4.24 зображено екран системи.



Записати клієнта

Клієнт(пошук за номером телефону) + К.П.

Марка та модель Номерний знак

Опис

ДД.ММ.ГГГГ

0 0

Закрити Зберегти

Рисунок 4.24 – Модальне вікно запису клієнта

На рис. 4.25-4.26 зображено екрани системи.

Записати клієнта

Богдан

Audi X3

AP85718AP

тест

29.04.2025

6 9

Закрити Зберегти

Рисунок 4.25 – Заповнене модальне вікно запису клієнта

29 квітня

6:00-9:00

Новий Послуги

Audi X3 - Богдан

тест

Закрити заявку

Рисунок 4.26 – Успішно доданий запис

Під час тестування було зроблено висновок, що функціонал модальних вікон працює як потрібно, але шрифт у модальному вікні також відрізняється.

4.4.3 Microsoft Edge

Перевіримо коректне відображення графічного інтерфейсу, валідацію полів, збереження токена авторизації та взаємодію з модальними вікнами з використанням браузеру Microsoft Edge. Перевірку коректного відображення графічного інтерфейсу та валідацію полів наведено на рис. 4.27-4.33. На рис. 4.27-4.28 зображено екрани системи.

AutoService CRM

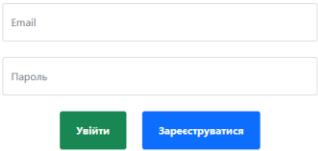


Рисунок 4.27 – Сторінка авторизації

AutoService CRM

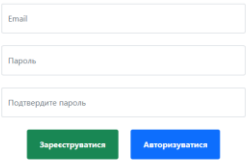
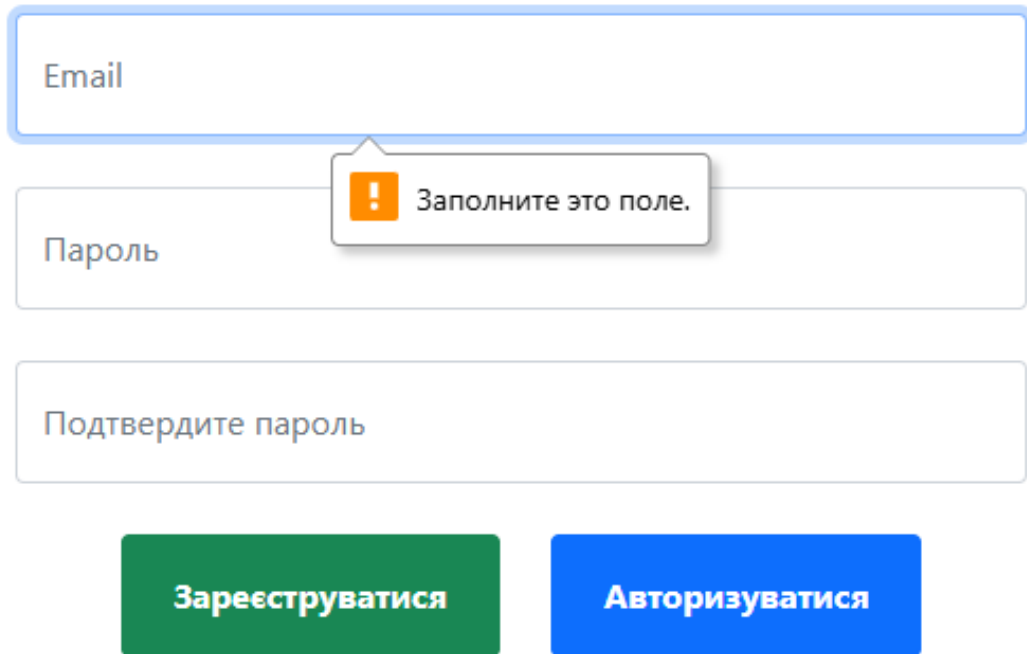


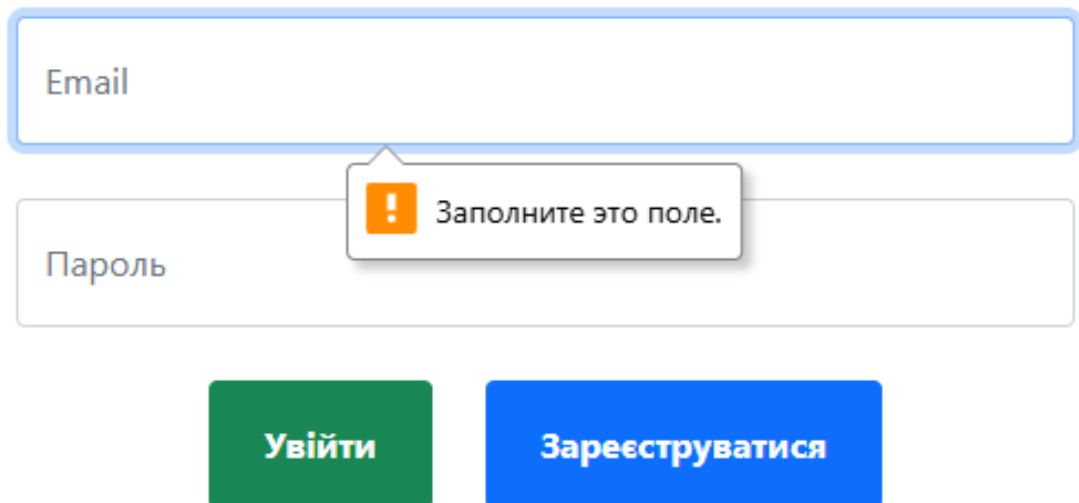
Рисунок 4.28 – Сторінка реєстрації

На рис. 4.29-4.30 зображено валідацію полів.



The image shows a registration form with three input fields: 'Email', 'Пароль' (Password), and 'Потвердите пароль' (Confirm password). The 'Email' field is highlighted with a blue border. A tooltip with an orange exclamation mark icon and the text 'Заполните это поле.' (Fill in this field.) points to the 'Пароль' field. Below the fields are two buttons: a green 'Зареєструватися' (Register) button and a blue 'Авторизуватися' (Log in) button.

Рисунок 4.29 – Попередження про незаповнене поле при реєстрації



The image shows a login form with two input fields: 'Email' and 'Пароль' (Password). The 'Email' field is highlighted with a blue border. A tooltip with an orange exclamation mark icon and the text 'Заполните это поле.' (Fill in this field.) points to the 'Пароль' field. Below the fields are two buttons: a green 'Увійти' (Log in) button and a blue 'Зареєструватися' (Register) button.

Рисунок 4.30 – Попередження про незаповнене поле при авторизації

На рис. 4.31-4.33 зображено екрани системи.

Запис

Запис

Клієнти

Налаштування

Вийти

Календар

Записати клієнта

20 квітня

12:00-15:00

Новий запис

Послуги

BMW X7 - Максим

asdsadas

Заміна колодок - 500

Разом: 500

Закрити заявку

29 квітня

6:00-9:00

Новий

Послуги

Audi X3 - Богдан

тест

Закрити заявку

< Попередній

Наступний >

Рисунок 4.31 – Сторінка записів

Клієнти

Запис

Клієнти

Налаштування

Вийти

Пошук за номером телефону

Додати клієнта

Ім'я	Номер(а) телефону	Авто	Персональна знижка
Максим	0957161032 0673664557	BMW X7 AP47581AP	10%
Богдан	0673664557	Audi X3 AP85718AP	5%
Лазарка	0958731257	Toyota Supra CPY228	15%

Рисунок 4.32 – Сторінка клієнтів

Налаштування

Запис

Клієнти

Налаштування

Вийти

Додавання нового статусу заявки

Новий статус заявки

Додати

Додавання нової послуги

Послуга

0

Додати

Рисунок 4.33 – Сторінка налаштувань

На рис. 4.34-4.36 изображено процес пошуку токену.

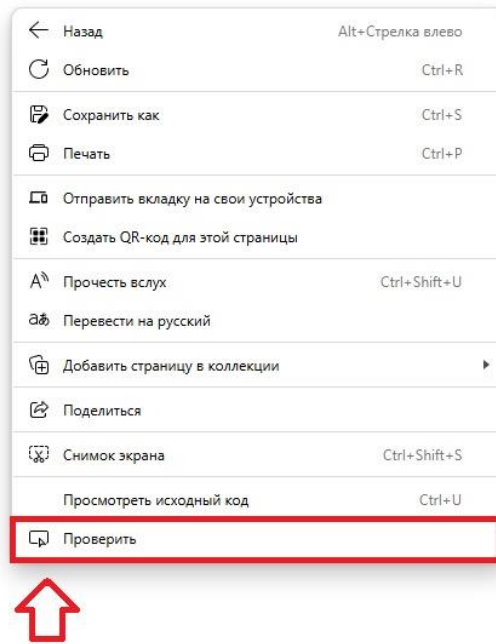


Рисунок 4.34 – Випадаюче меню

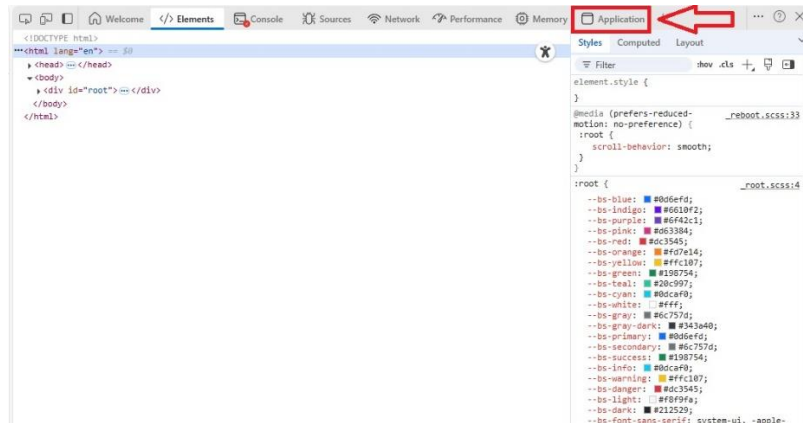


Рисунок 4.35 – Редактор коду

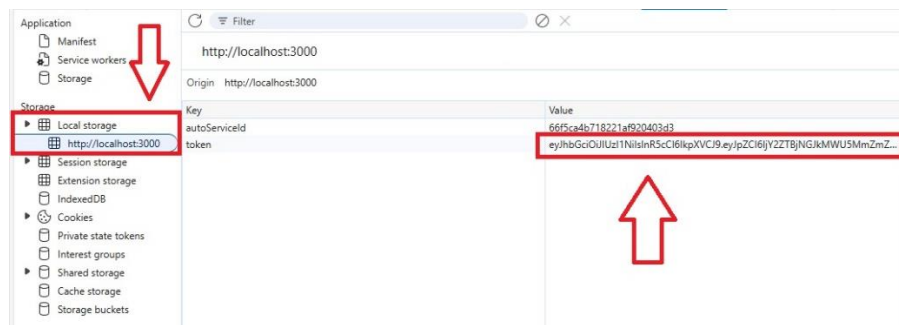
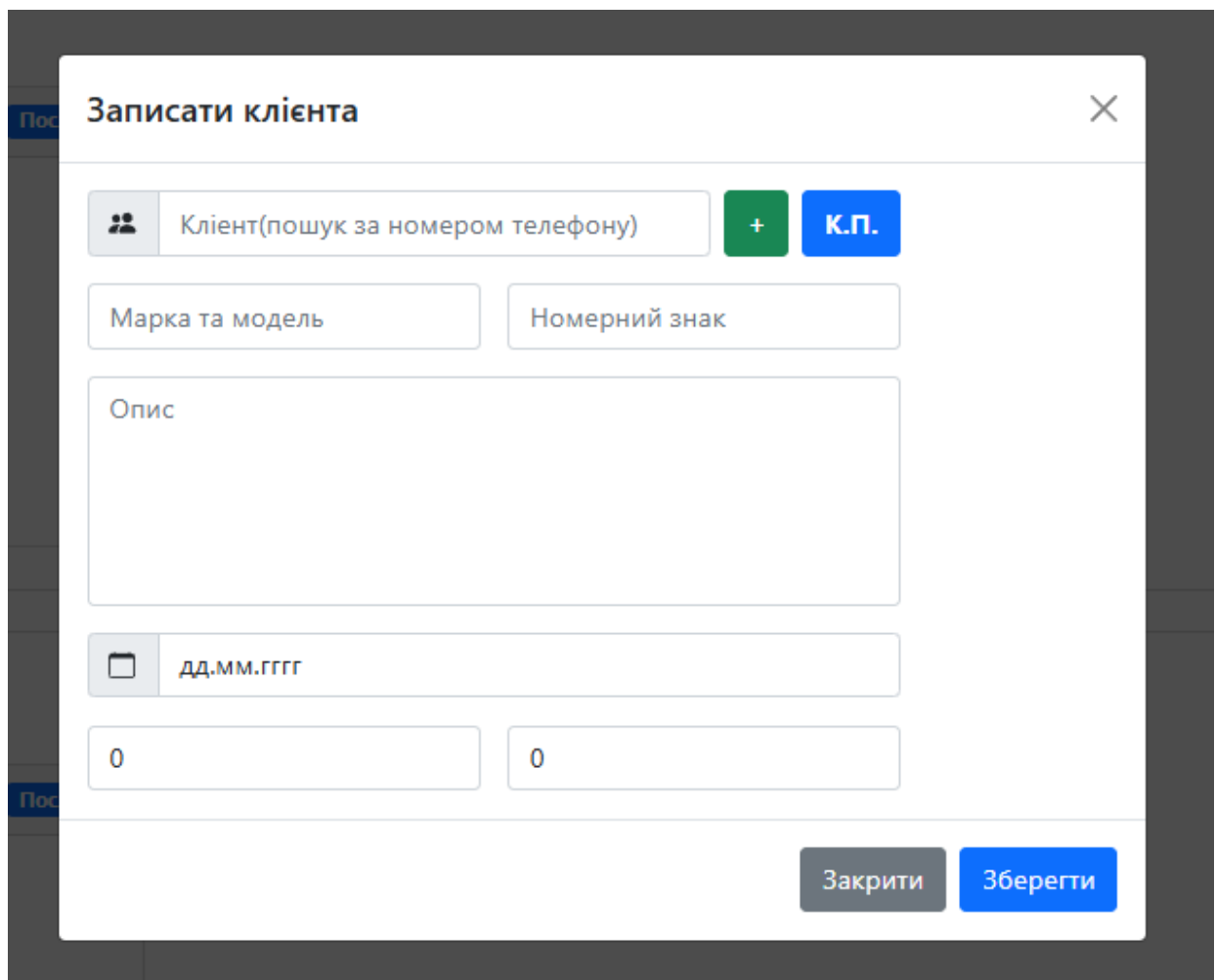


Рисунок 4.36 – Локальне сховище браузеру

У результаті тестування графічний інтерфейс та функціонал валідації продемонстрували очікувані результати та поведінку. Перейдемо до збереження токена. Для того щоб перевірити чи зберігається токен авторизації у браузері Microsoft Edge потрібно: натиснути правою кнопкою миші по екрану, обрати пункт «Перевірити», перейти у пункт меню «Application» та знайти там пункт Local storage. Увесь процес перевірки токена наведено на рис. 4.34-4.36

Токен коректно зберігається у локальному сховищі браузера Microsoft Edge. Залишилося перевірити правильну роботу модальних вікон. Перевірку правильного відображення модальних вікон зображено на рис. 4.37-4.39. На рис. 4.37 зображено екран системи.



Записати клієнта

Клієнт(пошук за номером телефону) + К.П.

Марка та модель

Номерний знак

Опис

ДД.ММ.ГГГГ

0 0

Закрити Зберегти

Рисунок 4.37 – Модальне вікно запису клієнту

На рис. 4.38-4.39 зображено екрани системи

Записати клієнта

Богдан

Audi X3

AP85718AP

тест

23.04.2025

12

15

Закрити

Зберегти

Рисунок 4.38 – Заповнене модальне вікно запису клієнта

23 квітня

12:00-15:00

Новий

Послуги

Audi X3 - Богдан

тест

Закрити заявку

Рисунок 4.39 – Успішно записаний клієнт

4.4.4 Результати тестування

Загальні результати тестування зведено у (табл. 4.1).

Таблиця 4.1 – Тестування системи

Критерій	Google Chrome	Mozilla Firefox	Microsoft Edge
Коректне відображення інтерфейсу	Інтерфейс відображається коректно	Інтерфейс відображається коректно, але деякі шрифти не жирні	Інтерфейс відображається коректно
Зберігання токenu	Токен зберігається у локальному сховищі	Токен зберігається у локальному сховищі	Токен зберігається у локальному сховищі
Валідація	Поля валідуються коректно	Поля валідуються коректно	Поля валідуються коректно
Функціонал модальних вікон	Модальні вікна функціонують згідно вимог	Модальні вікна функціонують згідно вимог	Модальні вікна функціонують згідно вимог

4.5 Висновки за отриманими результатами

Проведене тестування засвідчило коректну роботу та стабільність функціонування розробленої CRM-системи в актуальних версіях популярних веб-браузерів. Інтерфейс виявився адаптивним і зручним для користувача, що сприяє комфортній взаємодії з системою. Усі ключові можливості — авторизація, реєстрація, створення записів, використання модальних вікон — працюють відповідно до закладеної логіки та очікуваної поведінки.

ВИСНОВКИ

У процесі створення CRM-системи для станції технічного обслуговування було реалізовано повноцінну розробку як клієнтської, так і серверної частин застосунку. Попередньо проаналізовано особливості предметної області, визначено основні функціональні вимоги, а також обрано технології, які забезпечують стабільну, масштабовану й надійну роботу системи. Рішення дозволяє ефективно керувати заявками клієнтів, вести облік послуг, контролювати статус виконання робіт і підтримувати зручну базу даних клієнтів.

Фронтенд реалізований на базі React і Redux Toolkit, що дало змогу створити сучасний, інтуїтивно зрозумілий інтерфейс, оптимізований для швидкої та зручної роботи з даними. Завдяки продуманій архітектурі, з використанням кастомних хуків і централізованого керування станом, інтерфейс залишається легко підтримуваним і гнучким до змін. Особлива увага була приділена зручності користування — впроваджено модальні вікна, пагінацію, адаптивну навігацію та швидкий доступ до необхідної інформації.

Серверна частина побудована з використанням Node.js, Express та бази даних MongoDB, яка завдяки своїй документоорієнтованій структурі ідеально підходить для зберігання інформації про клієнтів, заявки та послуги. База даних логічно структурована на взаємопов'язані колекції, що сприяє зручному та цілісному управлінню інформацією.

У ході тестування в різних браузерях та перевірки ключових функцій було підтверджено стабільну роботу системи та відповідність її функціоналу заявленим вимогам. Реалізовані модулі авторизації, реєстрації, управління заявками та налаштувань успішно виконують покладені на них завдання, а користувацький інтерфейс забезпечує зручність роботи як для персоналу СТО, так і для адміністрації.

Таким чином, у результаті проєкту було створено сучасну, функціональну та масштабовану CRM-систему, яка дозволяє підвищити

ефективність роботи автосервісу, автоматизувати рутинні операції та надає зручні інструменти для ведення обліку клієнтів і наданих послуг.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. AutoRepair Cloud – Shop Management Software [Electronic resource]. – Access mode: <https://www.autorepaircloud.com/>.
2. Features [Electronic resource]. – Access mode: <https://www.autorepaircloud.com/features.html>.
3. What is Auto Repair Bill? [Electronic resource]. – Access mode: <https://www.capterra.com/p/133784/Auto-Repair-Bill/>.
4. Easy Auto Repair Shop Software [Electronic resource]. – Access mode: <https://www.autorepairbill.com/>.
5. How to abandon Russian programs in favor of Ukrainian CRM [Electronic resource]. – Access mode: <https://remonline.ua/ru/blog/ukrainska-crm/>.
6. Top 7 Frontend Frameworks to Use in 2025: Pro Advice [Electronic resource]. – Access mode: <https://roadmap.sh/frontend/frameworks>.
7. Порівнюємо React, Angular і Vue — найпопулярніші бібліотеки й фреймворки у 2022 році [Electronic resource]. – Access mode: <https://dou.ua/forums/topic/39933/>.
8. Why to Use Angular in 2024: An Overview of Angular Pros and Cons [Electronic resource]. – Access mode: <https://leobit.com/blog/why-to-use-angular-in-an-overview-of-angular-pros-and-cons/>.
9. Pros and Cons of the Vue.js Framework [Electronic resource]. – Access mode: <https://naturally.com/blog/pros-cons-vue-js>.
10. Best Backend Frameworks for 2025: A Developer's Guide to Making the Right Choice [Electronic resource]. – Access mode: <https://dev.to/developerbishwas/best-backend-frameworks-for-2025-a-developers-guide-to-making-the-right-choice-45i0>.
11. Advantages and Disadvantages of ExpressJS [Electronic resource]. – Access mode: <https://data-flair.training/blogs/expressjs-advantages-and-disadvantages/>.
12. What Is the Django Framework? The Complete Beginner's Guide

[Electronic resource]. – Access mode: <https://careerfoundry.com/en/blog/web-development/django-framework-guide/>.

13. Response body [Electronic resource]. – Access mode: <https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/>.

14. Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others [Electronic resource]. – Access mode: <https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb>.

15. PostgreSQL Advantages and Disadvantages [Electronic resource]. – Access mode: <https://www.aalpha.net/blog/pros-and-cons-of-using-postgresql-for-application-development/>.

16. SQLite Advantages [Electronic resource]. – Access mode: <https://www.tpointtech.com/sqlite-advantages-and-disadvantages>.

17. Redux Toolkit. The official, opinionated, batteries-included toolset for efficient Redux development [Electronic resource]. – Access mode: <https://redux-toolkit.js.org/>.

ДОДАТОК А
Технічне завдання

Вступ

Цей документ є технічним завданням на розробку програмного забезпечення — CRM-системи для станції технічного обслуговування (СТО). Основна мета створення даного застосунку полягає в автоматизації процесів обліку клієнтів, записів на обслуговування, ведення інформації про надані послуги, а також у покращенні координації роботи між працівниками СТО.

Система повинна мати зручний, інтуїтивно зрозумілий інтерфейс, забезпечувати швидкий доступ до необхідних даних, гарантувати безпеку інформації та стабільну роботу. Програмне забезпечення розробляється з урахуванням сучасних стандартів веб-розробки і реалізується як клієнт-серверне рішення.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмна реалізація CRM-системи для станції технічного обслуговування», затверджене наказом Національного університету «Запорізька політехніка» № 209 від 28 квітня 2025 р.

A.2 Основні вимоги до програми, що розроблюється

Для гарантування коректного, стабільного й ефективного функціонування програмного забезпечення важливо чітко окреслити основні вимоги, яким повинна відповідати система протягом усього життєвого циклу. Ці вимоги охоплюють як функціональні характеристики, так і вимоги до дизайну інтерфейсу, надійності, а також технічні умови розгортання та експлуатації. Їх формування базується на реальних потребах станції технічного обслуговування та сучасних підходах до створення інформаційних систем.

У цьому розділі представлені основні вимоги до розроблюваного

програмного продукту, виходячи з його головної мети — автоматизації обліку клієнтів, управління записами та послугами, а також оптимізації внутрішніх робочих процесів. Для зручності подальшої реалізації, тестування та впровадження системи, вимоги класифіковано за логічними групами.

A.2.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати реалізацію наступного функціоналу:

- безпечну реєстрацію та авторизацію користувачів із відповідним захистом персональних даних;
- створення та видалення заявок на обслуговування;
- ведення бази клієнтів з можливістю збереження контактної інформації, відомостей про автомобіль та історії обслуговування;
- автоматичний розрахунок вартості послуг для клієнтів;
- сортування, фільтрацію та пагінацію списку заявок для зручності навігації;
- налаштування довідників послуг та статусів заявок, доступне адміністраторам системи;
- відображення графіку записів у форматі зручних візуальних карток із розподілом по днях;
- можливість роботи з кількома відділеннями автосервісу через систему облікових записів.

A.2.2 Вимоги до інтерфейсу програми

Користувацький інтерфейс має бути простим для розуміння, візуально привабливим і адаптованим для роботи на різних пристроях. Перемикання між основними розділами повинно здійснюватися через зручне бічне меню або

верхню навігаційну панель із чітким позначенням активної сторінки. Усі форми повинні бути логічно організованими, а елементи керування — добре помітними та комфортними для взаємодії.

A.2.3 Вимоги до надійності

Програмне забезпечення повинно гарантувати збереження даних у випадку втрати з'єднання або перезапуску системи. Уся інформація має зберігатися у базі даних MongoDB із належним рівнем захисту доступу. Автентифікація користувачів повинна реалізовуватись за допомогою JWT-токенів. Архітектура застосунку має передбачати незалежність його компонентів, що забезпечить можливість оновлення або модифікації окремих частин без ризику для стабільності всієї системи.

A.2.4 Вимоги до складу та параметрів технічних засобів

Для ефективного використання програми на клієнтському боці користувачеві рекомендується мати комп'ютер або ноутбук із наступними мінімальними технічними характеристиками:

- процесор: не нижче Intel Core i3 (8-го покоління) або AMD Ryzen 3;
- оперативна пам'ять (RAM): не менше 4 Гб (рекомендується 8 Гб);
- накопичувач: SSD для швидкого запуску браузера та системи;
- операційна система: Windows 10/11, macOS 10.15+ або актуальні версії Linux;
- браузер: останні стабільні версії Google Chrome, Mozilla Firefox або Microsoft Edge;
- роздільна здатність екрану: мінімум 1366×768 (рекомендується Full HD 1920×1080);
- підключення до Інтернету: стабільне з'єднання зі швидкістю від 10

Мбіт/с для належного обміну даними з сервером.

Ці характеристики забезпечують швидке завантаження інтерфейсу, стабільну взаємодію з серверною частиною та комфортну роботу без зависань і затримок.

Для адміністраторів з розширеними правами (наприклад, при роботі з великими обсягами даних або редагуванні бази даних) рекомендується використовувати комп'ютери з потужнішими характеристиками, зокрема, з 8 ГБ оперативної пам'яті та процесором не нижче Intel Core i5.

ДОДАТОК Б
Текст програми

Б.1 Файл app.js

```

app.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    console.log(user);
    if (user === null || !(await bcrypt.compare(password, user.password))) {
      return res.status(400).json({ message: "Неправильний логін або пароль" });
    }
    const autoServiceId = user.autoServiceId;
    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
      expiresIn: "1h",
    });
    res.json({
      token,
      autoServiceId,
      user: { id: user._id, email: user.email },
    });
  } catch (error) {
    res.status(500).json({ message: "Ошибка серверу" });
  }
});

app.post("/checkAuth", async (req, res) => {
  let token;
  console.log("Authorization header:", req.headers.authorization);
  if (
    req.headers.authorization &&
    req.headers.authorization.startsWith("Bearer")
  ) {
    token = req.headers.authorization.split(" ")[1];
  }

  if (!token) {
    return res.status(401).json({ message: "Немає токєну" });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    const user = await User.findById(req.user.id).select("-password");
    const { email, id } = user;
    res.json({ message: "Доступ дозволено", user: { id: id, email: email } });
  }
});

```

```

    } catch (err) {
      return res.status(401).json({ message: "Неверный токен" });
    }
  });

app.post("/reg", async (req, res) => {
  const email = req.body.email;
  const password = req.body.password;
  const user = await User.findOne({ email: email });

  if (user) {
    res.status(400).json({ message: "Користувач вже існує!" });
  } else {
    const hashedPass = await bcrypt.hash(password, salt);

    const newUser = new User({
      email: email,
      password: hashedPass,
    });

    await newUser.save();

    res.status(200).json({ message: "Користувач успішно зареєстрований" });
  }
});

app.post("/changeStatus", async (req, res) => {
  try {
    const autoServiceId = req.body.autoServiceId;
    const date = new Date(req.body.day);
    const recordId = req.body.recordId;
    const status = req.body.status;
    const autoService = await AutoService.findOne({ _id: autoServiceId });
    if (autoService) {
      const dayIndex = autoService.days.findIndex(
        (day) => new Date(day.dayDate).getTime() === date.getTime()
      );
      console.log(dayIndex);
      if (dayIndex !== -1) {
        const record = autoService.days[dayIndex].records.id(recordId);
        if (record) {
          record.status = status;
          autoService.days[dayIndex].records = autoService.days[
            dayIndex
          ].records.filter((r) => r.status !== "Закрит");
        }
      }
    }
  } catch (err) {
    res.status(500).json({ message: "Внутрішня помилка сервера" });
  }
});

```

```

    autoService.days = autoService.days.filter(
      (day) => day.records.length > 0
    );

    autoService.save().then(() => {
      res.status(200).json(autoService.days);
    });
  } else {
    res.status(400).json({ message: "Запис не знайдено!" });
  }
  } else {
    res.status(400).json({ message: "День не знайдено!" });
  }
  } else {
    res.status(400).json({ message: "Автосервис не знайдено!" });
  }
} catch (error) {
  res.status(401).json({ message: "Помилка на сервері" });
}
});
app.post("/addRecord", async (req, res) => {
  try {
    const autoServiceId = req.body.autoServiceId;
    const client = req.body.client;
    const clientId = req.body.clientId;
    const car = req.body.car;
    const carNumber = req.body.carNumber;
    const description = req.body.description;
    const date = new Date(req.body.date);
    const { from, to } = req.body.duration;
    const status = req.body.status;

    const autoService = await AutoService.findOne({ _id: autoServiceId });

    const newRecord = new Record({
      clientName: client,
      clientId: clientId,
      car: car,
      carNumber: carNumber,
      description: description,
      date: date,
      duration: {
        from: from,
        to: to,

```

```

    },
    status: status,
  });

const dayIndex = autoService.days.findIndex(
  (day) => new Date(day.dayDate).getTime() === date.getTime()
);

console.log(dayIndex);

if (dayIndex !== -1) {
  autoService.days[dayIndex].records.push(newRecord);
} else {
  autoService.days.push({ dayDate: new Date(date), records: newRecord });
}

autoService.days.sort(
  (a, b) => new Date(a.dayDate).getTime() - new Date(b.dayDate).getTime()
);

autoService.save().then(() => {
  res.status(200).json(autoService.days);
});
} catch (error) {
  res.status(500).json({ message: "Помилка на сервері!" });
}
});

app.post("/addClient", async (req, res) => {
  try {
    console.log(req.body);
    const autoServiceId = req.body.autoServiceId;
    const name = req.body.name;
    const brand = req.body.car.brand;
    const model = req.body.car.model;
    const numbers = req.body.car.number;
    const phoneNumbers = req.body.phoneNumber;
    const personalDiscount = req.body.personalDiscount;

    const autoService = await AutoService.findOne({ _id: autoServiceId });

    if (autoService) {
      const newClient = new Client({
        name: name,
        phoneNumber: phoneNumbers,

```

```

    car: {
      brand: brand,
      model: model,
      number: numbers,
    },
    personalDiscount: personalDiscount,
  });
  autoService.clients.push(newClient);

  await autoService.save();
  res.status(200).json(autoService.clients);
} else {
  res.status(401).json({ message: "Автосервис не знайдено" });
}
} catch (error) {
  res.status(500).json({ message: "Помилка на сервері!" });
}
});
app.post("/updateClient", async (req, res) => {
  try {
    const autoServiceId = req.body.autoServiceId;
    const clientId = req.body._id;
    console.log(autoServiceId, clientId);
    const name = req.body.name;
    const brand = req.body.car.brand;
    const model = req.body.car.model;
    const numbers = req.body.car.number;
    const phoneNumbers = req.body.phoneNumber;
    const personalDiscount = req.body.personalDiscount;

    const autoService = await AutoService.findOne({ _id: autoServiceId });

    if (autoService) {
      const client = await autoService.clients.id(clientId);
      if (client) {
        client.name = name;
        client.phoneNumber = phoneNumbers;
        client.car = {
          brand: brand,
          model: model,
          number: numbers,
        };
        client.personalDiscount = personalDiscount;

```

```

    await autoService.save();
    res.status(200).json(autoService.clients);
  } else {
    res.status(401).json("Клиєнта не знайдено!");
  }
} else {
  res.status(401).json({ message: "Автосервис не знайдено" });
}
} catch (error) {
  res.status(500).json({ message: "Помилка на сервері!" });
}
});
app.post("/addStatusRecord", async (req, res) => {
  try {
    const autoServiceId = req.body.autoServiceId;
    const newOption = req.body.status;
    const autoService = await AutoService.findOne({ _id: autoServiceId });

    if (autoService) {
      const statusWorkOptions = await autoService.options.statusWorkOptions;

      if (statusWorkOptions) {
        statusWorkOptions.push(newOption);

        await autoService.save();
        res.status(200).json(autoService.options);
      } else {
        res.status(401).json({ message: "Не знайдено!" });
      }
    } else {
      res.status(401).json({ message: "Автосервис не знайдено!" });
    }
  } catch (error) {
    res.status(500).json({ message: "Помилка на сервері" });
  }
});
app.post("/addService", async (req, res) => {
  const autoServiceId = req.body.autoServiceId;
  const newService = {
    service: req.body.service.serviceName,
    cost: req.body.service.cost,
  };
  const autoService = await AutoService.findOne({ _id: autoServiceId });

```

```

if (autoService) {
  const serviceOptions = await autoService.options.servicesOptions;
  console.log(serviceOptions);
  if (serviceOptions) {
    serviceOptions.push(newService);

    await autoService.save();
    res.status(200).json(autoService.options);
  } else {
    res.status(401).json({ message: "Не знайдено!" });
  }
} else {
  res.status(401).json({ message: "Автосервис не знайдено!" });
}
});
app.post("/insertAutoservice/:name", async (req, res) => {
  try {
    const autoService = new AutoService({ name: req.params.name });
    await autoService.save();

    console.log("AutoService successfully added");
    res
      .status(201)
      .json({ message: "AutoService successfully added", data: autoService });
  } catch (error) {
    console.error("Error adding AutoService:", error);
    res
      .status(500)
      .json({ message: "Failed to add AutoService", error: error.message });
  }
});

```

Б.2 Файл Options.tsx

```

const handleAddStatus = (statusRecord: string) => {
  if (statusRecord !== "") {
    const creds: NewStatusCreds = {
      status: statusRecord,
      autoServiceId: localStorage.getItem("autoServiceId"),
    };
    dispatch(addStatusRecord(creds))
      .unwrap()
      .catch((err) => {

```

```

        alert(err);
    });
    setStatusRecord("");
    alert("Успішно додано новий статус!");
} else {
    alert("Заповніть усі поля!");
}
};

const handleAddService = (service: string, cost: number) => {
    console.log(service, cost);
    if (service !== "" && cost !== 0) {
        const creds: NewServiceCreds = {
            service: {
                serviceName: service,
                cost: cost,
            },
            autoServiceId: localStorage.getItem("autoServiceId"),
        };

        dispatch(addService(creds))
            .unwrap()
            .catch((err) => {
                alert(err);
            });
        setService("");
        setServiceCost(0);
        alert("Успішно додана нова послуга!");
    } else {
        alert("Заповніть усі поля!");
    }
};

```

Б.3 Файл RecordsContent.tsx

```

const limitedDays = days && days.slice(0, 10);
const handleNext = () => {
    if (days && (currentPage + 1) * itemsPerPage < days.length) {
        setCurrentPage(currentPage + 1);
    }
};

const handlePrevious = () => {

```

```

    if (currentPage > 0) {
      setCurrentPage(currentPage - 1);
    }
  };

  const startIndex = currentPage * itemsPerPage;
  const endIndex = startIndex + itemsPerPage;
  const displayedDays =
    limitedDays && limitedDays.slice(startIndex, endIndex);

  return (
    <div className="content d-flex flex-column gap-4 mb-4">
      {displayedDays &&
        displayedDays.map((day) => {
          return <Day day={day} />;
        })}
      <RecordsFooter
        onNext={handleNext}
        onPrevious={handlePrevious}
        hasNext={days && endIndex < days?.length}
        hasPrevious={currentPage > 0}
      />
    </div>
  );

```

Б.4 Файл useClientsModal.ts

```

function handleSubmit() {
  if (
    !name ||
    !mark ||
    !model ||
    !numbers ||
    !phonePrimary ||
    personalDiscount < 0
  ) {
    alert("Заповніть усі поля!");
  } else if (!client) {
    const credentials: Client = {
      _id: null,
      name: name,
      car: {
        brand: mark,

```

```

    model: model,
    number: numbers,
  },
  phoneNumber: [phonePrimary, phoneSecondary],
  personalDiscount: personalDiscount,
  autoServiceId: localStorage.getItem("autoServiceId"),
};

dispatch(addClient(credentials)).catch((err) => {
  alert(err);
});
toggle();
} else {
  const credentials: Client = {
    _id: client._id,
    name: name,
    car: {
      brand: mark,
      model: model,
      number: numbers,
    },
    phoneNumber: [phonePrimary, phoneSecondary],
    personalDiscount: personalDiscount,
    autoServiceId: localStorage.getItem("autoServiceId"),
  };

  dispatch(updateClient(credentials)).catch((err) => {
    alert(err);
  });
  toggle();
}
}

```

ДОДАТОК В
Слайди презентації

Програмна реалізація CRM-системи для станції технічного обслуговування.

Виконав:
студент групи КНТ-221
Керівник:
к.т.н. доцент

Максим СУР
Євгеній ГОФМАН

Рисунок В.1 – Слайд 1

Об'єкт предмет та мета роботи

2

Об'єкт дослідження – процес автоматизації управління заявками та клієнтами станції технічного обслуговування (СТО) за допомогою CRM-системи.

Предмет дослідження – методи та технології розробки CRM-системи для СТО, що включають роботу з базою даних клієнтів, управління заявками та послугами. Для реалізації було використано сучасний стек технологій: React, React Toolkit, Express.js, MongoDB, JWT tokens, bcrypt, axios, react-router-dom.

Мета дослідження – створення програмного забезпечення, яке допоможе автоматизувати запис клієнтів, управління послугами та статусами заявок. Це дозволить покращити роботу СТО, зробить процеси швидшими та зручнішими як для менеджерів, так і для клієнтів.

Рисунок В.2 – Слайд 2

Постановка завдання

3

Метою роботи є дослідити можливості створення CRM-системи для СТО, спрямованої на автоматизацію бізнес-процесів, покращення управління заявками та клієнтською базою, а також забезпечення безпечної взаємодії між користувачами.

Завдання дослідження:

- Дослідити потреби СТО у сфері управління клієнтами та заявками;
- Проаналізувати існуючі CRM-рішення для вибору оптимального підходу;
- Визначити вимоги до майбутньої системи;
- Обґрунтувати вибір технологій для фронтенду та бекенду;
- Спроекувати архітектуру та моделі системи;
- Реалізувати інтерфейс користувача та бізнес-логіку;
- Провести тестування та оцінити ефективність рішення;

Рисунок В.3 – Слайд 3

Порівняння існуючих аналогів

4

Критерій	AutoRepair Cloud	Auto Repair Bill	RemOnline
Основні функції	Управління клієнтами, записи на обслуговування, облік запчастин	Виставлення рахунків, контроль витрат, ведення клієнтської бази	Облік клієнтів, управління записами, фінансовий облік, аналітика
Вартість	\$34.99 на місяць	\$200 на одне робоче місце	від €29 на місяць
Гнучкість налаштувань	Середня	Низька	Висока
Інтеграція	Платіжні системи	Бухгалтерський облік	Онлайн-оплати, сторонні сервіси
Тип доступу	Хмарний	Десктопний	Хмарний
Легкість впровадження	Висока	Середня	Висока
Кастомізація	Низька	Низька	Середня

Рисунок В.4 – Слайд 4

Порівняння фреймворків для фронтенду

5

Критерій	React	Angular	Vue
Поріг входу	Середній	Високий	Низький
Підтримка TypeScript	Є, за бажанням	Обов'язкова	Є, але не обов'язкова
Продуктивність	Висока	Висока	Висока
Гнучкість	Висока	Середня	Висока
Розмір збірки	Малий	Великий	Малий
Документація	Дуже добра	Добра, але складна	Чітка і зрозуміла
Спільнота	Дуже велика	Велика, але менш активна	Середня
Легкість інтеграції	Висока	Середня	Висока
Критерій	React	Angular	Vue

Рисунок В.5 – Слайд 5

Порівняння фреймворків для бекенду

6

Характеристика	ExpressJS	Django
Мова програмування	JavaScript (Node.js)	Python
Архітектура	Мінімалістичний, без чіткої структури	MVC (Model-View-Controller), сувора структура
Поріг входу	Помірний. Добре підходить тим, хто вже знайомий з JavaScript	Вбудовані, включаючи підтримку WebSockets, мікросервісів та GraphQL
Безпека	Потребує ручної реалізації механізмів захисту	Має вбудовані засоби захисту від CSRF, XSS, SQL-ін'єкцій та ін.
Вбудований ORM	Немає. Потрібно використовувати сторонні (Sequelize, Mongoose тощо)	Є потужний вбудований ORM
Масштабованість	Підходить для мікросервісної архітектури, масштабування вручну	Підходить для великих проєктів, але масштабування складніше
Гнучкість	Дуже гнучкий. Можна з нуля формувати будь-яку архітектуру	Менш гнучкий, багато рішень прийняті за замовчуванням
Продуктивність	Висока, підходить для додатків у реальному часі	Середня. Може поступатися в продуктивності Express.js у високонавантажених системах

Рисунок В.6 – Слайд 6

Порівняння баз даних

7

Критерій	MongoDB	PostgreSQL	SQLite
Тип бази даних	NoSQL, документна база даних	Реляційна база даних	Легка, файлова реляційна база даних
Продуктивність	Висока продуктивність при великих обсягах даних	Висока продуктивність, оптимізація запитів	Швидкість запису та читання в 35% швидше за файлові системи
Масштабованість	Горизонтальна масштабованість через шардінг	Вертикальна масштабованість, підтримка реплікацій	Обмежена масштабованість
Інсталяція	Вимагає налаштування серверів і додаткових компонентів	Потрібно встановити на сервері або локально	Не потребує інсталяції, легко інтегрується
Гнучкість схеми	Без схеми, динамічні документи	Строга схема, підтримка складних типів даних	Простота в схемі, підтримка невеликих змін
Безпека	Підтримка автентифікації, авторизації, шифрування	Розширена безпека на рівні ролей і привілеїв	Параметри безпеки на рівні ОС та додатків
Транзакції	Підтримуються	Обмежена підтримка	Підтримуються

Рисунок В.7 – Слайд 7

Схеми основних алгоритмів

8



Рисунок В.8 – Слайд 8

Файлова структура компонентів клієнтської частини

9

```

components
├── Auth
│   ├── Auth.tsx
│   └── Registration.tsx
├── Clients
│   ├── Clients.tsx
│   ├── ClientsModal.tsx
│   ├── ControlPage
│   │   └── ControlPage.tsx
│   ├── Header
│   │   └── Header.tsx
│   ├── List
│   │   └── List.tsx
│   ├── Options
│   │   └── Options.tsx
│   ├── Records
│   │   ├── Cards
│   │   │   ├── Day.tsx
│   │   │   └── Record.tsx
│   │   ├── Modal
│   │   │   ├── Modal.tsx
│   │   │   ├── ModalContent.tsx
│   │   │   └── ModalHeader.tsx
│   │   ├── Records.tsx
│   │   ├── RecordsContent.tsx
│   │   ├── RecordsFooter.tsx
│   │   └── RecordsHeader.tsx

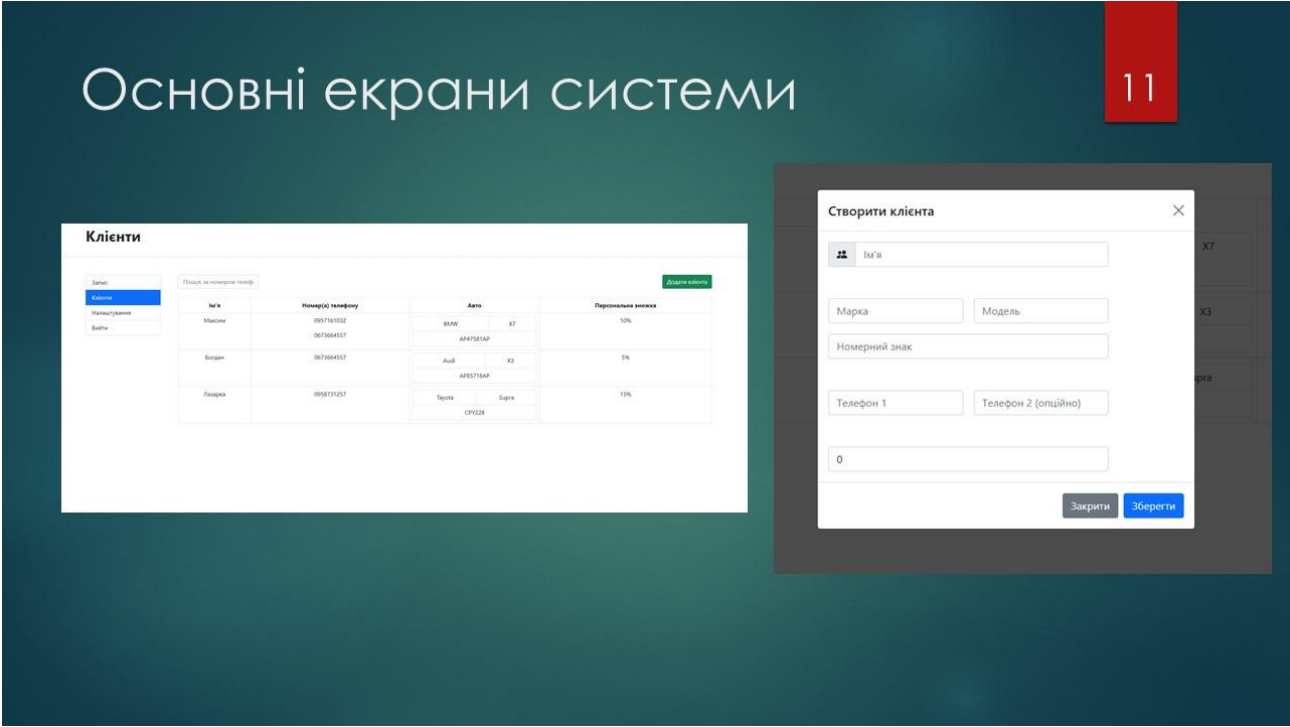
```

Рисунок В.9 – Слайд 9

Основні екрани системи

10

Рисунок В.10 – Слайд 10



Створити клієнта

Ім'я

Марка

Модель

Номерний знак

Телефон 1

Телефон 2 (опційно)

0

Закрити

Зберегти

Рисунок В.11 – Слайд 11

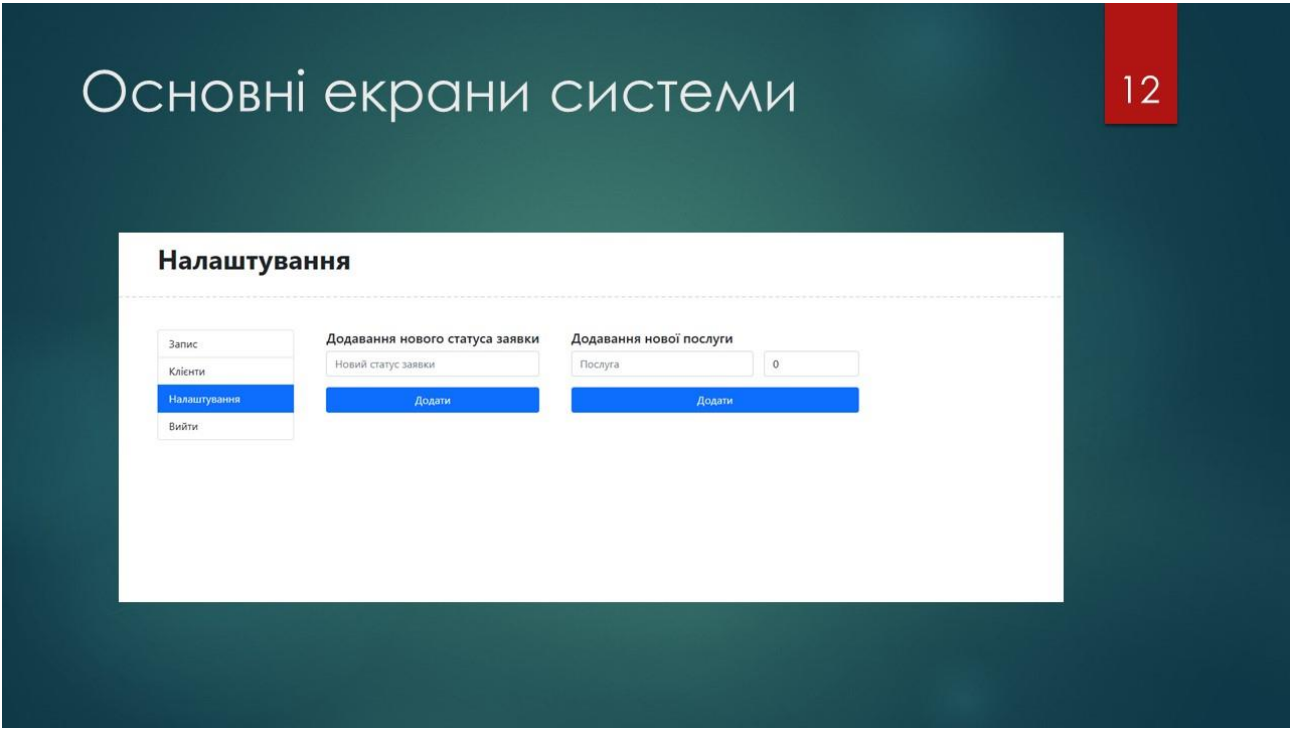


Рисунок В.12 – Слайд 12

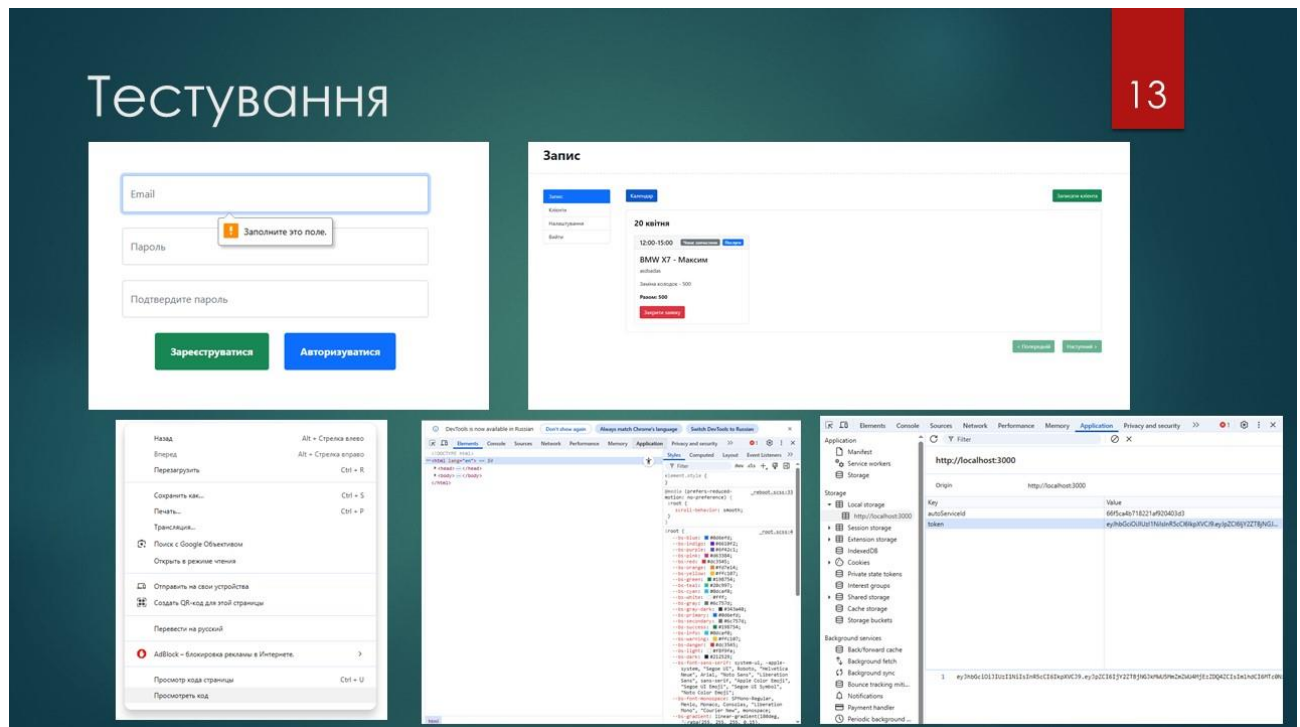


Рисунок В.13 – Слайд 13

Результати тестування			
Критерій	Google Chrome	Mozilla Firefox	Microsoft Edge
Коректне відображення інтерфейсу	Інтерфейс відображається коректно	Інтерфейс відображається коректно, але деякі шрифти не жирні	Інтерфейс відображається коректно
Зберігання токена	Токен зберігається у локальному сховищі	Токен зберігається у локальному сховищі	Токен зберігається у локальному сховищі
Валідація	Поля валідуються коректно	Поля валідуються коректно	Поля валідуються коректно
Функціонал модальних вікон	Модальні вікна функціонують згідно вимог	Модальні вікна функціонують згідно вимог	Модальні вікна функціонують згідно вимог

Рисунок В.14 – Слайд 14

Висновки

15

У ході виконання дипломної роботи бакалавра було реалізовано низку ключових завдань, спрямованих на створення ефективного програмного рішення для автоматизації процесів станції технічного обслуговування.

Зокрема:

- досліджено потреби СТО щодо обліку клієнтів і управління заявками;
- проаналізовано існуючі CRM-рішення для виявлення актуальних підходів до побудови подібних систем;
- визначено основні функціональні та нефункціональні вимоги до системи;
- обґрунтовано вибір оптимальних технологій для фронтенд- і бекенд-розробки;
- спроектовано архітектуру, реалізацію моделей;
- створено сучасний інтерфейсу користувача;
- протестовано функціональні можливості і оцінку результатів.

Усі поставлені завдання були успішно виконані в повному обсязі.

Створена система відповідає заявленим вимогам, забезпечує стабільну роботу, має зручний інтерфейс та є придатною для подальшого впровадження й використання в умовах реального автосервісу.

Варто зазначити, що система має потенціал для подальшого вдосконалення — зокрема, шляхом розширення функціоналу, інтеграції з мобільними платформами або сторонніми сервісами, а також впровадження аналітики для моніторингу ефективності роботи сервісу.

Рисунок В.15 – Слайд 15