НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Вебзастосунок для ведення обліку технічного стану транспортних засобів»

Виконав:
студент IV курсу, групи КП-03
Довженко Роман Олександрович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Старший викладач кафедри ПМА ФПМ
Мальчиков Володимир Вікторович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2023 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Довженко Роману Олександровичу

1. Тема проєкту «Вебзастосунок для ведення обліку технічного стану транспортних засобів», керівник проекту Заболотня Тетяна Миколаївна, к.т.н., доцент, затверджені наказом по університету від «30» травня 2024 р. №2205-с.

2. Термін подання студентом проєкту «13» червня 2024 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - огляд існуючих програмних рішень;
   - обґрунтування вибору засобів розроблення;
   - структурно-алгоритмічна організація застосунку;
   - особливості реалізації програмного забезпечення;

5. Перелік обов'язкового графічного матеріалу:
   - структурна схема бази даних (креслення);
   - структура схема вебзастосунку (креслення);
   - структурна схема контейнерів серверної частини вебзастосунку (плакат);
   - блок-схема алгоритму обчислення графіку витрат з лінією тренду (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2023 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	13.11.2023	
2.	Розроблення та узгодження технічного завдання	22.11.2023	
3.	Розроблення структури вебзастосунку	15.12.2023	
4.	Підготовка матеріалів першого розділу дипломного проєкту	28.12.2023	
5.	Розроблення дизайну сторінок та графічних елементів	01.02.2024	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2024	
7.	Програмна реалізація вебзастосунку	11.03.2024	
8.	Тестування вебзастосунку	18.03.2024	
9.	Підготовка матеріалів третього розділу дипломного проєкту	27.03.2024	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	15.04.2024	
11.	Підготовка графічної частини дипломного проєкту	22.04.2024	
12.	Оформлення документації дипломного проєкту	27.05.2024	

Студент Роман ДОВЖЕНКО

Керівник проєкту Тетяна ЗАБОЛОТНЯ

АНОТАЦІЯ

Даний дипломний проєкт присвячений створенню вебзастосунку для ведення обліку технічного обслуговування транспортних засобів. Основна задача проєкту – забезпечити потреби відповідальних власників транспортних засобів у сучасних інструментах для контролю та збереження історії технічного обслуговування, що сприятиме підвищенню безпеки експлуатації та прозорості при купівлі-продажу транспортних засобів.

Програма реалізована в формі вебсайту для власників транспортних засобів та працівників станцій технічного обслуговування, призначеного для моніторингу статистики витрат, додавання нагадувань про необхідність проведення технічного обслуговування, передачі інформації наступному власнику за необхідності, а також ведення записів стосовно певних маніпуляцій над агрегатами транспортного засобу.

У даному дипломному проєкті розглянуто існуючі на ринку рішення, їх переваги та недоліки, технології для реалізації серверної та клієнтської частини застосунку. Описано функціональні та нефункціональні вимоги до програмного забезпечення. Представлено структурну та алгоритмічну організацію системи. Реалізовано серверну та клієнтську частини застосунку. Проведено тестування системи, яке підтверджує готовність проєкту до використання.

Впровадження даного програмного застосунку сприятиме підвищенню рівня безпеки для відповідальних власників транспортних засобів, а також підвищуватиме рівень довіри на ринку продажу.

ABSTRACT

This diploma project is dedicated to the creation of a web application for tracking the maintenance of vehicles. The main objective of the project is to meet the needs of responsible vehicle owners with modern tools for monitoring and maintaining the history of maintenance, which will contribute to increased operational safety and transparency in the buying and selling of vehicles.

The program is implemented in the form of a website for vehicle owners and service station workers, designed for monitoring expense statistics, adding reminders about the need for maintenance, transferring information to the next owner if necessary, and keeping records of specific manipulations on vehicle components.

In this diploma project, existing market solutions, their advantages and disadvantages, and technologies for implementing the server and client parts of the application are reviewed. Functional and non-functional requirements for the software are described. The structural and algorithmic organization of the system is presented. The server and client parts of the application are implemented. System testing has been conducted, confirming the readiness of the project for use.

The implementation of this software application will contribute to increased safety for responsible vehicle owners and will also enhance trust in the sales market.

ДП.045440-01-90. Вебзастосунок для ведення обліку технічного стану транспортних засобів. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Вебзастосунок для	5	
	ведення обліку технічного		
	стану транспортних		
	засобів. Технічне		
	завдання		
ДП.045440-03-81	Вебзастосунок для	78	
	ведення обліку технічного		
	стану транспортних		
	засобів. Пояснювальна		
	записка		
ДП.045440-04-51	Вебзастосунок для	4	
	ведення обліку технічного		
	стану транспортних		
	засобів. Програма та		
	методика тестування		
ДП.045440-05-34	Вебзастосунок для	15	
	ведення обліку технічного		
	стану транспортних		
	засобів. Керівництво		
	користувача		
ДП.045440-06-99	Вебзастосунок для	1	
	ведення обліку технічного		
	стану транспортних		
	засобів. Структурна схема		
	бази даних. ER-діаграма.		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-07-99	Вебзастосунок для	1	
	ведення обліку технічного		
	стану транспортних		
	засобів. Структурна схема		
	вебзастосунку.		
	UML-діаграма.		
ДП.045440-08-98	Вебзастосунок для	1	
	ведення обліку технічного		
	стану транспортних		
	засобів. Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2023 р.

ВЕБЗАСТОСУНОК ДЛЯ ВЕДЕННЯ ОБЛІКУ ТЕХНІЧНОГО СТАНУ ТРАНСПОРТНИХ ЗАСОБІВ

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Роман ДОВЖЕНКО

2023

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Вебзастосунок для ведення обліку технічного стану транспортних засобів.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання відповідальними власниками транспортних засобів та працівниками станцій технічного обслуговування з метою збереження історії технічного обслуговування транспортних засобів, включаючи заміну деталей, ремонтні роботи, планові огляди та інші технічні маніпуляції. Програмне забезпечення призначене для встановлення нагадувань про необхідні процедури та підвищення довіри при продажу автомобіля шляхом верифікації виконаних робіт.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Вебзастосунок повинен забезпечувати такі основні функції:

1) можливість зберігати записи про технічне обслуговування;
2) можливість додавання, завершення та видалення запланованих подій;
3) можливість передачі історії ведення технічного обслуговування наступному власнику;

4) надсилання нагадувань про необхідність проведення технічного обслуговування.

Додаткові вимоги:

1) зберігання паролів у захешованому вигляді;
2) відображення календаря з подіями;
3) генерація статистики витрат;
4) можливість динамічного оновлення вмісту вебсторінок.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   - «Структурна схема бази даних»;
   - «Структура схема вебзастосунку».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи……..…..16.11.2023

Розроблення та узгодження технічного завдання…….....27.11.2023

Розроблення структури вебзастосунку…….....15.12.2023

Розроблення дизайну сторінок та графічних елементів..…..05.02.2024

Програмна реалізація вебзастосунку.…….……...15.03.2024

Тестування вебзастосунку.……..…....08.04.2024

Підготовка матеріалів текстової частини проєкту……29.04.2024

Підготовка матеріалів графічної частини проєкту……..…15.05.2024

Оформлення технічної документації проєкту..……..…28.05.2024

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ ВЕДЕННЯ ОБЛІКУ ТЕХНІЧНОГО СТАНУ ТРАНСПОРТНИХ ЗАСОБІВ

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Роман ДОВЖЕНКО

2024

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ТО – технічне обслуговування;

СТО – станція технічного обслуговування;

HTTP – Hypertext Transfer Protocol (протокол передачі гіпертексту, основний протокол для обміну даними в Інтернеті);

REST – Representational State Transfer (представлення стану представлення, архітектурний стиль для розробки мережевих застосунків);

RESTful – спосіб організації мережевого застосунку, який дотримується принципів REST (представлення стану представлення);

API – Application Programming Interface (набір правил і інструкцій для взаємодії програм та компонентів програм);

JVM – Java Virtual Machine (віртуальна машина Java, середовище для виконання Java-програм);

ООП – об'єктно-орієнтоване програмування (парадигма програмування, що базується на концепції "об'єктів", які містять дані та методи);

JIT – Just-in-Time (компіляція, процес перетворення коду програми на машинний код під час виконання програми);

JDBC – Java Database Connectivity (API для взаємодії Java-програм з реляційними базами даних);

GIL – Global Interpreter Lock (глобальний інтерпретаторний блок, механізм для управління доступом до об'єктів з багатьох потоків виконання в Python);

CLR – Common Language Runtime (загальна середовище виконання, віртуальна машина для виконання програм на мовах .NET);

АОП – аспектно-орієнтоване програмування (парадигма програмування для відокремлення додаткових функціональностей від основної бізнес-логіки);

RPC – Remote Procedure Call (видалений виклик процедур, механізм для виклику функцій на віддаленому комп'ютері);

MVC – Model-View-Controller (модель-представлення-контролер, архітектурний шаблон для розробки програмного забезпечення);

XML – Extensible Markup Language (розширюваний мова розмітки, мова для створення документів з структурованими даними);

JMS – Java Message Service (служби повідомлень Java, API для створення, відправлення, отримання та обробки повідомлень в Java-програмах);

JPA – Java Persistence API (API для управління персистентними об'єктами в Java-програмах);

JTA – Java Transaction API (API для керування транзакціями в Java-програмах);

JAR – Java Archive (файл, формат для зберігання компільованих класів та ресурсів Java);

ORM – Object-Relational Mapping (відображення об'єктно-реляційного відображення, технологія для відображення об'єктів в базі даних на об'єкти в програмному коді);

SQL – Structured Query Language (мова структурованих запитів, мова програмування для керування реляційними базами даних);

YAML – YAML Ain't Markup Language (мову розмітки, людино-читабельний формат обміну даними);

JSON – JavaScript Object Notation (формат обміну даними, заснований на синтаксисі JavaScript);

БД – база даних;

ELK – Elasticsearch, Logstash, Kibana (стек програмного забезпечення для пошуку, візуалізації та аналізу журналів);

DOM – Document Object Model (модель об'єктів документа, структура HTML або XML документа як дерево об'єктів);

JSX – JavaScript Extension (розширення JavaScript, що дозволяє декларативно створювати компоненти користувальницького інтерфейсу);

MVVM – Model-View-ViewModel (модель-вид-в'язка, архітектурний шаблон для розробки програмного забезпечення);

CLI – Command Line Interface (інтерфейс командного рядка, спосіб взаємодії з комп'ютером за допомогою введення текстових команд);

СУБД – система управління базами даних (програмне забезпечення для керування базами даних);

E2E – End-to-End (кінцевий до кінцевого, підхід у тестуванні, який охоплює весь процес від початку до кінця);

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення, основні операції з даними);

ER – Entity-Relationship (сутність-відношення, модель для опису даних у базах даних);

IoC – Inversion of Control (інверсія управління, принцип проектування для передачі контролю від програми до контейнера);

URL – Uniform Resource Locator (уніфікований вказівник ресурсу, адреса для доступу до ресурсів в Інтернеті);

PDF – Portable Document Format (портативний формат документів, формат файлів для представлення документів незалежно від програмного забезпечення);

XLSX – Excel Spreadsheet (формат файлів для збереження електронних таблиць Microsoft Excel);

SMS – Short Message Service (служба коротких повідомлень, технологія для передачі коротких текстових повідомлень).

ВСТУП

В епоху цифрових технологій все більше раніше звичних для людей речей стають доступними в будь-якому куточку світу, перш за все завдяки мобільним телефонам та відповідним вебзастосункам. Однією з галузей, в якій спеціальні застосунки мають великий попит, є автомобільна індустрія.

Від моменту винайдення автомобілі були переважно механічними пристроями, і власникам доводилось знаходити і лагодити певні несправності, покладаючись на свій досвід. Трохи пізніше для автомобілів наступного покоління почали випускати інструкції користувача. Це були багатосторінкові журнали, в яких були описані певні вузли автомобіля, оскільки вони мають термін експлуатації, і їх треба змінювати задля збереження безпеки водія та пасажирів. Чітке знання про те, коли було проведено останнє технічне обслуговування, дозволяє вчасно виявити потенційні проблеми з автомобілем, які можуть призвести до аварійної ситуації.

Інформація про технічний стан автомобіля також є важливою для потенційних покупців. Потенційні покупці часто перевіряють історію технічного обслуговування перед покупкою. Раніше власники вели історію цих змін, переважно у зошитах чи спеціальних щоденниках. У сучасному світі власники автомобілів можуть скористатися вебзастосунками для ведення детальної історії технічного обслуговування.

Серед наявних на ринку рішень поки не існує такого, що б дозволяло передавати історію обслуговування покупцю автомобіля та мало б систему верифікації виконання роботи сервісним центром зі збереженням документів.

Даний дипломний проєкт присвячений розробленню програмного забезпечення, яке б забезпечувало збереження та передачу при необхідності історії технічного обслуговування автомобіля, а також реалізовувало функцію верифікації та збереження документів виконання робіт з

обслуговування автомобіля сервісними центрами. Розроблений в рамках виконання даного дипломного проєкту застосунок покращує якість обслуговування автомобілів шляхом збереження історії ТО. Нагадування про необхідність проведення планових робіт дозволяє власникам автомобілів бути впевненими, що вони завжди вчасно проводять необхідні процедури обслуговування, що важливо для забезпечення безпеки експлуатації та продовження терміну служби автомобіля. Можливість верифікації та збереження документів щодо виконаних робіт сервісними центрами, сприяє довірі між учасниками ринку автотранспорту.

1. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

1.1. Основні положення та аспекти предметної області

В контексті зростання цифрових технологій і їх впливу на різні галузі автомобільна індустрія не залишається осторонь. Перехід до електронних систем зберігання і передачі інформації стосовно історії ТО транспортного засобу є логічним кроком для оптимізації процесів обслуговування та покращення зв’язку між власниками, покупцями автомобілів і сервісними центрами.

Аналіз основних проблем, з якими стикаються різні категорії користувачів, каже про те, що відповідальним власникам автомобілів складно переконати потенційного покупця у достовірності виконання певних технічних маніпуляцій стосовно транспортного засобу, таких як заміна чи ремонт певних вузлів, регулярна заміна технічних рідин, проходження планового технічного огляду. Існуючі на ринку рішення дозволяють вести журнал ТО, але не мають системи верифікації проведення обслуговування, таким чином, для потенційного покупця постає питання стосовно достовірності інформації, через що він змушений прибігти до використання сторонньої допомоги, і, відповідно, надлишкових витрат при підборі автомобіля.

Введення системи верифікації виконання роботи сервісним центром є одним зі шляхів вирішення проблем власників та потенційних покупців автомобілів. Використання такої системи сервісним центром є ефективним способом заохочення звертатись саме до цього сервісного центру, а не до конкурентів.

Система верифікації ТО передбачає збереження документів виконання робіт, фотографій заміни певних вузлів і особливу відмітку про те, що ця операція над транспортним засобом підтверджена сервісним центром, в якому вона проводилась.

1.2. Опис існуючих рішень

Даний параграф дипломного проєкту присвячений вивченню існуючих рішень, огляду їх функціональних можливостей, програмних реалізацій, переваг та недоліків. Визначення слабких та сильних сторін допоможе краще зрозуміти потреби ринку, а також дозволить виявити можливі напрямки для подальшого вдосконалення розроблюваної системи.

1.2.1. Simply Auto

Simply Auto – інструмент для обліку даних про автомобіль для Android та iPhone, який допомагає вести облік і планування технічного обслуговування та ремонтів, а також контролювати витрати на автомобіль [1].

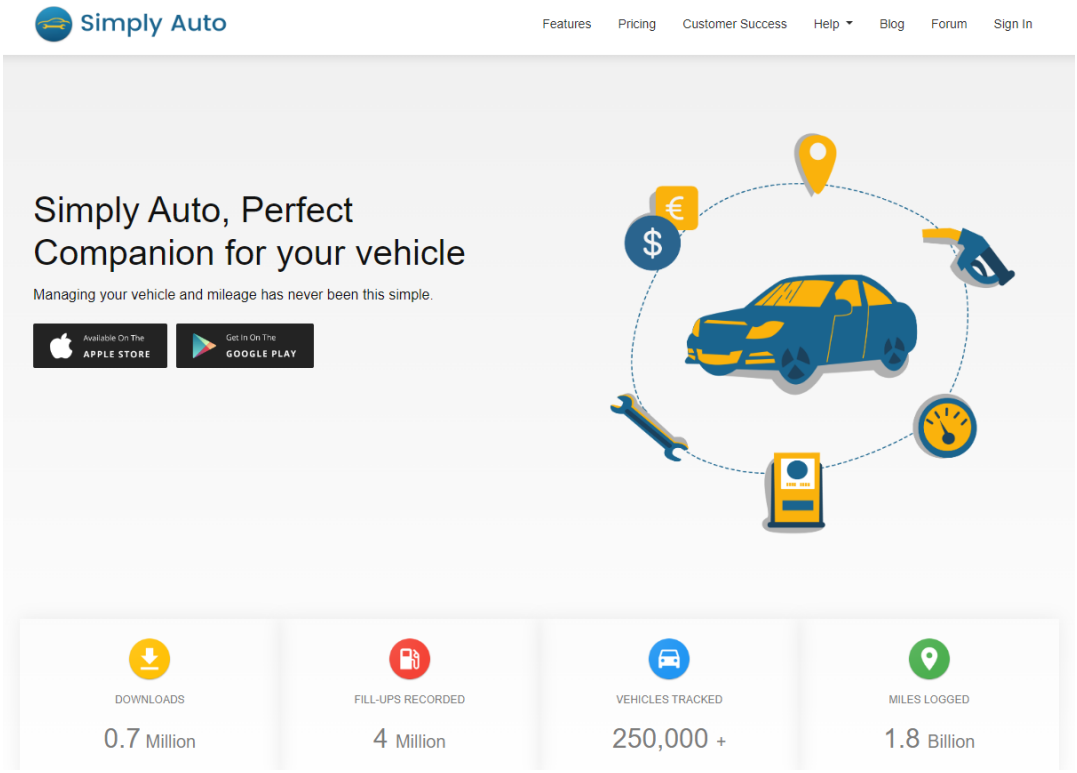


Рис. 1.1. Скріншот головної сторінки сайту для застосунку Simply Auto

Серед функціональних можливостей застосунку можна виділити:

- планування ТО автомобіля, встановлення нагадування по даті чи за пробігом;

- запис заправок, пробігу, поїздок та витрат;
- збереження чеку до запису ТО або заправки;
- надсилання звіту про заправки та пробіг на електронну пошту;
- генерація статистики витрат;
- прикладання чеку;
- одночасна підтримка декількох профілів автомобілів;
- можливість позначати поїздки як приватні та службові для подальшого встановлення тарифу для різних типів поїздок.

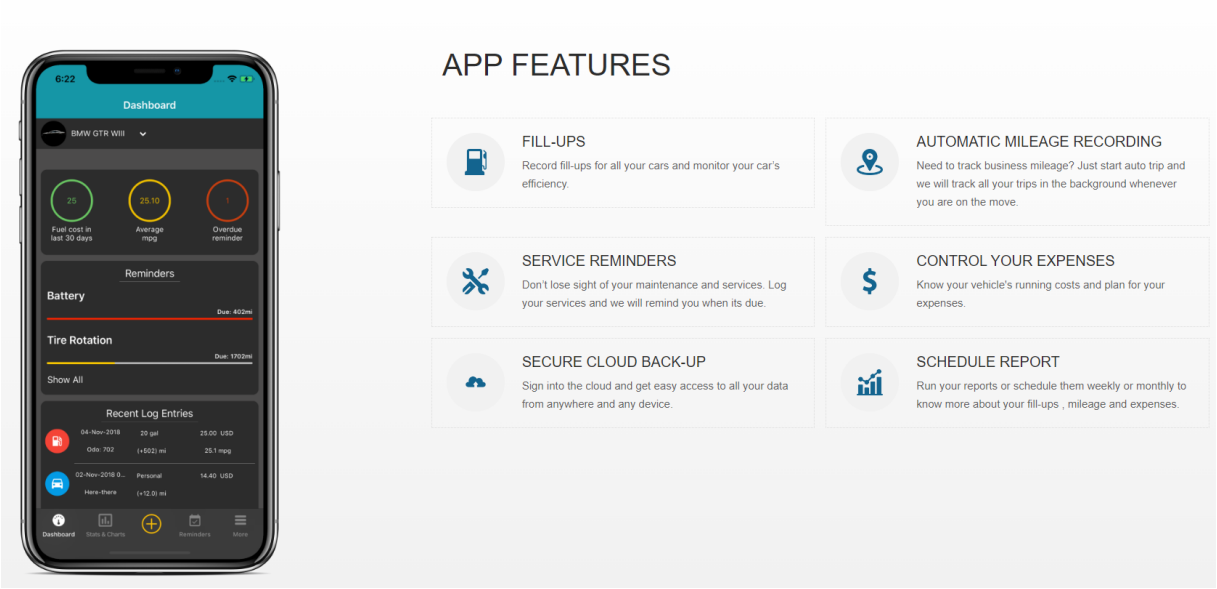


Рис. 1.2. Скріншот переліку функціональних можливостей застосунку Simply Auto

Основна перевага приведеного програмного застосунку - це широкий спектр функціональних можливостей.

Серед недоліків Simply Auto можна виділити наступні:

- даний застосунок доступний лише для пристроїв на базі Android та iOS;
- відсутність системи верифікації проведення роботи;
- відсутня українська локалізація;
- неможливо передати дані історію ТО автомобіля;

- застарілий дизайн;
- відсутня підтримка для iPhone (на момент 18.04.2024 за інформацією App Store останнє оновлення для платформи iOS було 24.03.2022).

1.2.2. CarDiary

CarDiary – кросплатформний застосунок для ведення журналу обслуговування автомобіля. За допомогою цього застосунку можна зберігати дані про ремонти, заправки, подорожі та оплату податків. Надає можливість відслідковувати витрати за останній тиждень, місяць або рік [2].

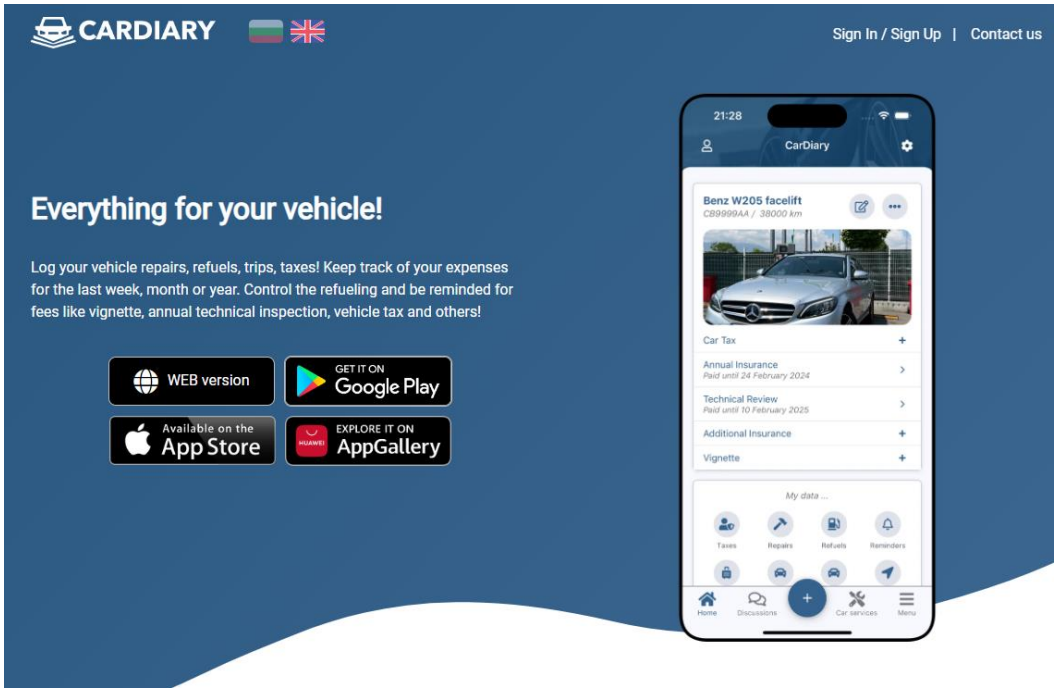


Рис. 1.3. Скріншот головної сторінки сайту для застосунку CarDiary

Серед переваг CarDiary як інструменту для ведення журналу історії автомобіля можна виділити:

- можливість генерації даних у форматі PDF про історію автомобіля;
- кросплатформеність;
- сучасний, повністю адаптивний дизайн;
- відсутність реклами;
- можливість переглянути найближчі автосервіси;

- інтеграція з Google Calendar;
- регулярне оновлення та підтримка застосунку.

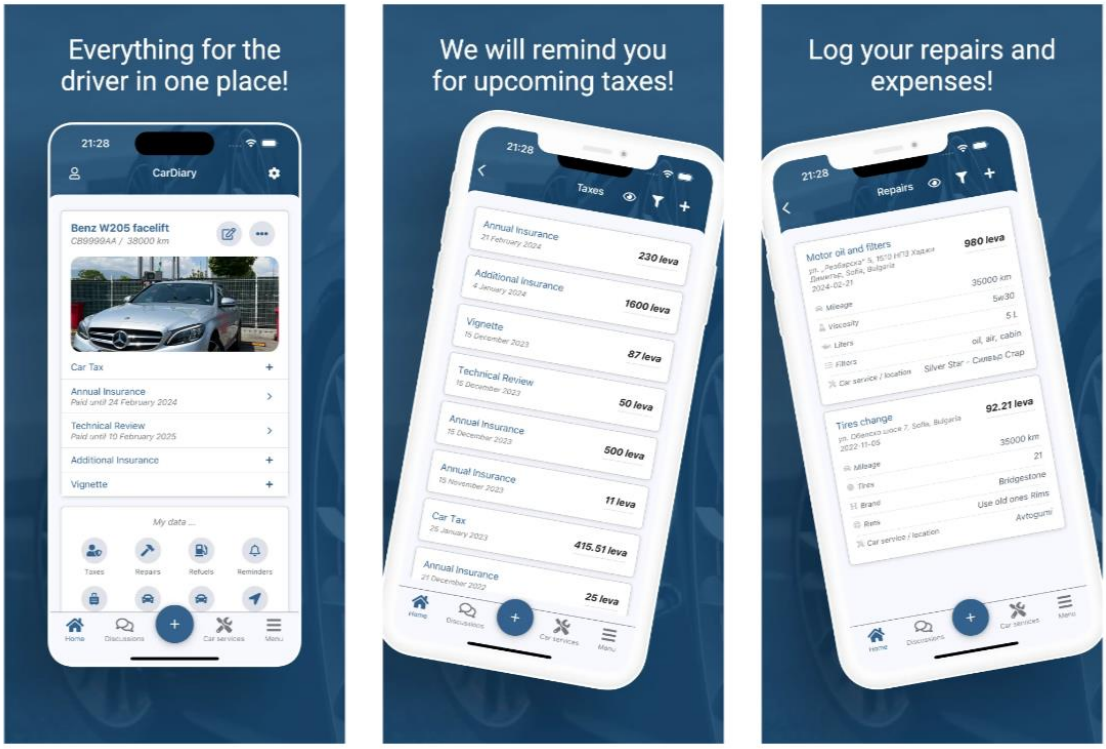


Рис. 1.4. Ілюстрація декількох вікон застосунку CarDiary

В порівнянні з попереднім, даний застосунок є більш зручним для пересічного користувача, проте має ряд недоліків:

- відсутність можливості додати документи та чеки виконання роботи;
- неможливість передавати історію ТО;
- відсутність системи верифікації проведення роботи;
- відсутність української локалізації.

1.2.3. myCarLog

Наступним для аналізу був обраний ще один інструмент для контролю витрат на автомобіль – myCarLog. На відміну від попередників, даний програмний застосунок спеціалізований тільки для пристроїв на платформі iOS [3].

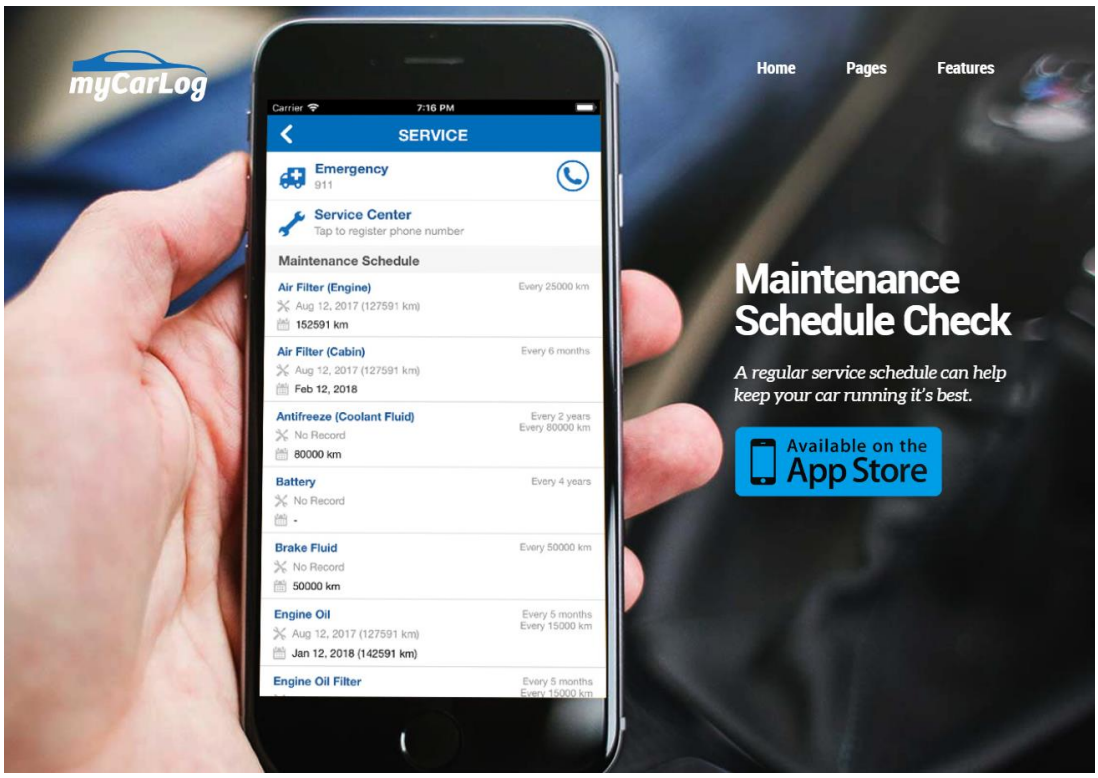


Рис. 1.5. Скріншот головної сторінки сайту для застосунку myCarLog

На сайті розробника для даного додатку описані такі функціональні можливості:

- ведення історії заправок, обслуговування та інших витрат;
- управління інформацією щодо кількох машин;
- генерування річних звітів;
- спостереження за своєчасним виконанням регламентних робіт;
- зберігання номерів аварійних служб;
- інтеграція з голосовим асистентом “Siri” для додавання даних щодо заправки, витрат та обслуговування.

В цілому, застосунок myCarLog пропонує корисні функції, проте в порівнянні з іншими аналогічними додатками має обмежену функціональність. Найбільшим недоліком myCarLog є обмеження його працездатності однією платформою, що може викликати необхідність вибору аналогічного додатку для користувачів, які володіють пристроями на іншій платформі. Однак, варто зазначити, що серед проаналізованих

раніше застосунків тільки цей додаток має інтеграцію з голосовим асистентом.

1.3. Порівняльний аналіз існуючих рішень

В результаті проведеного аналізу вдалось визначити основні переваги та недоліки існуючих на ринку рішень. Нижче наведена порівняльна таблиця функціональних можливостей розглянутих застосунків.

Таблиця 1.1

Порівняння функціональних можливостей існуючих програмних рішень

Функціональні можливості \ Застосунок	Simply Car	CarDiary	myCarLog
Ведення історії обслуговування та інших витрат	+	+	+
Планування ТО	+	+	+
Нагадування про необхідність проведення ТО	+	+	+
Підтримка декількох автомобілів	+	+	+
Кросплатформеність	+/–	+	–
Генерація статистики	+	+	+
Можливість зберегти чеки, фотографії, документи про виконання роботи	+/–	–	–
Верифікація проведення ТО	–	–	–
Можливість передавати історію ТО	–	–	–

Порівняльна таблиця демонструє, що кожне з існуючих на ринку рішень має свої сильні та слабкі сторони. Безумовно, всі вони виконують

свою основну функцію, а саме ведення журналу обслуговування автомобіля, проте, існують деякі спільні проблеми.

Перш за все, жодне з існуючих рішень не має системи верифікації виконання роботи з можливістю зберігання чеків, фотографій та документів виконання роботи, яка гарантує, що дані про обслуговування автомобіля, введені користувачем, є достовірними, і забезпечує уникнення помилок та маніпуляцій з даними про технічний стан автомобіля.

Більшість існуючих рішень обмежена платформою, що знижує їх доступність для користувачів, які використовують пристрої на несумісній з додатком платформі.

Два з трьох застосунків мають застарілий дизайн, що призводить до негативного враження від продукту, навіть якщо його функціональність різноманітною. Естетичний та інтуїтивно зрозумілий дизайн впливає на сприйняття користувачем продукту і є частиною задоволення від користування додатком.

Загальною проблемою для всіх проаналізованих додатків є неможливість передавати дані про історію автомобіля наступному власнику та відсутність української локалізації.

1.4. Висновки до розділу 1

Під час написання першого розділу дипломного проєкту проведено аналіз існуючих рішень, що присутні на ринку. Під час огляду їх функціональних можливостей та програмних реалізацій вдалося виявити ключові аспекти, які впливають на задоволення потреб користувачів.

Дані про функціональність були зіставлені в порівняльній таблиці. Отримана інформація дозволила прийти до висновку, що наявні рішення мають обмеження функціональності та не забезпечують повного задоволення потреб користувачів, а отже, існує певний простір для створення нового застосунку, який, враховуючи накопичений досвід, поєднуватиме переваги та усуне недоліки вже існуючих рішень.

2. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБЛЕННЯ

При проєктуванні та реалізації вебзастосунків існує декілька архітектурних підходів, серед яких одним з найпоширеніших і ефективних є підхід з використанням клієнт-серверної архітектури. В рамках розроблення даного вебзастосунку для ведення обліку технічного стану транспортних засобів обрано дворівневу клієнт-серверну архітектуру з міркувань ефективності, безпеки та масштабованості.

Клієнт-серверна архітектура передбачає поділ обов’язків між клієнтською та серверною частинами програми. Це дозволяє оптимізувати роботу застосунку шляхом розподілення навантаження між клієнтом та сервером. Сервер в такому випадку виступає як централізований ресурс, який керує доступом до даних та забезпечує їхню цілісність. Це допомагає забезпечити однорідність даних та уникнути конфліктів у одночасній роботі декількох клієнтів. Крім того, централізований сервер може забезпечити автентифікацію та авторизацію користувачів, що знижує ризик несанкціонованого доступу.

Зв'язок між клієнтською та серверною частиною в рамках даного дипломного проєкту буде забезпечуватися за допомогою стандартного для мережі Інтернет протоколу комунікації HTTP з використанням архітектурного стилю REST. Використання RESTful API спрощує розроблення та підтримку вебзастосунків, оскільки дозволяє забезпечити чітку роздільність між клієнтською та серверною частинами, а також спрощує інтеграцію з іншими системами. Такий підхід сприяє підвищенню масштабованості застосунку, оскільки дозволяє легко додавати нові функціональні можливості, а також робить його більш доступним та привабливим для інших сервісів і додатків для взаємодії та інтеграції. Інші сервіси при необхідності зможуть легко взаємодіяти з застосунком, використовуючи HTTP-запити. Загальний підхід до розроблення на основі

RESTful API дозволить застосунку бути частиною більш широкої екосистеми і є загальноприйнятим стандартом.

2.1. Вибір архітектури серверної частини

При виборі технологій для реалізації серверної частини додатку слід відштовхуватися від вибору серверної архітектури. Оскільки розроблювана система повинна мати функціональність нагадування про необхідність проведення ТО, то постає питання про необхідність забезпечення високої відмовостійкісті.

Вибір архітектурного шаблону, який буде використовуватися для серверної частини, стоїть між монолітною або мікросервісною архітектурою. Розглянемо їх нижче.

Монолітна архітектура – підхід до створення програмного забезпечення, за якого всі компоненти застосунку зосереджені в одній кодовій базі та працюють як один процес. Серед переваг цієї архітектури очевидно можна виділити такі як:

- знаходження всієї кодової бази в одному місці дозволяє значно прискорити швидкість розроблення;
- запуск, оновлення вже існуючих і додавання нових функціональних можливостей вимагає мінімальних зусиль;
- простота розгортання;
- полегшення процесу моніторингу та відслідковування помилок.

Проте, ця архітектура не позбавлена недоліків:

- при збільшенні об’єму кодової бази суттєво збільшується час на додавання змін, оскільки компоненти жорстко між собою зв’язані;
- по мірі збільшення монолітного застосунку виникають проблеми з його масштабуванням та гнучкістю;
- у разі виходу з ладу одного з компонентів виникають проблеми у функціонуванні всього застосунку.

Мікросервісна архітектура – методологія розроблення програмного забезпечення, за якого монолітна система розбивається на невеликі сервіси. Кожен з таких сервісів відповідає за свою частину бізнес логіки, і має обмежену функціональність. Взаємодія між декількома мікросервісами відбувається через програмний інтерфейс застосунку (API), завдяки чому досягається висока автономність мікросервісів. Основними плюсами мікросервісної архітектури є:

- можливість працювати над розробленням декількох мікросервісів паралельно;
- висока відмовостійкість, яка забезпечує мінімальний вплив збоїв в одному з мікросервісів на роботу всієї системи;
- можливість прив’язати до кожного мікросервіса свою базу даних, дозволяючи кожному сервісу керувати консистентністю та структурою бази даних, що дозволяє можна використовувати для однієї категорії сутностей в системі як реляційні, так і нереляційні бази даних;
- поліпшена масштабованість.

Серед мінусів мікросервісної архітектури можна виділити такі:

- мікросервісна архітектура – це, перш за все, рішення для складних і масштабних проєктів, оскільки процес розроблення вимагає уважності в плануванні системи;
- складнощі в процесі моніторингу та відслідковуванню помилок;
- висока складність налаштування та розгортання мережі;
- можливі проблеми, пов’язані з версійністю і оновленням сервісів.

Отже, при виборі між монолітною та мікросервісною архітектурою краще віддати перевагу мікросервісній архітектурі. Відповідно до мікросервісної архітектури, в розроблюваній системі слід передбачити окремий сервіс, який буде займатися надсиланням повідомлень про необхідність проведення ТО. У разі високого навантаження на цей сервіс, при виході його з ладу, інші компоненти системи залишаться працюючими.

Оскільки основна функціональність розроблюваної системи це надсилання повідомлень про необхідність проведення ТО, то слід враховувати те, що можливе високе навантаження. Таким чином, для ефективної роботи застосунку варто передбачити можливість його масштабування. В даному випадку під цим формулюванням мається на увазі вертикальне або горизонтальне масштабування.

Давайте розглянемо їх:

- вертикальним називається масштабування, при якому збільшення продуктивності програмного забезпечення виконується шляхом підвищення обчислювальної потужності системи;
- горизонтальним називається масштабування, при якому відбувається розподіл структурних компонентів програмного забезпечення серед окремих фізичних машин, що призводить до підвищення продуктивності за рахунок збільшення кількості серверів, що паралельно виконують одні й ті ж операції.

Обрана мікросервісна архітектура дозволяє горизонтально масштабувати сервіси, тим самим розподіляти між ними навантаження.

Наступним архітектурним рішенням в рамках даного дипломного проєкту, буде скомбінувати мікросервісну архітектуру з подійно-орієнтованою.

Подійно-орієнтована архітектура – концепція розроблення програмного забезпечення, яка базується на подійно-орієнтованому підході, основана ідея якого полягає в розділені системи на незалежні компоненти, які взаємодіють між собою через події. Це дозволяє створювати розподілені системи, які легко масштабуються і володіють високою відмовостійкістю. Комбінація такої архітектури з мікросервісною робить сервісну взаємодію асинхронною і фактично прибирає будь-яку зв’язність між компонентами системи.

В якості засобу реалізації подійно-орієнтованої архітектури, а саме шини для передачі подій між сервісами, було розглянуто декілька популярних систем повідомлень:

- ActiveMQ і RabbitMQ – системи повідомлень, створені для корпоративних задач, тобто забезпечують надійні гарантії доставки повідомлень. За замовчування, вони налаштовані на підтвердження кожного повідомлення і оснащені вбудованими механізмами відновлення, такими як черги мертвих повідомлень.
- Kafka – це платформа потокової передачі подій. Має перевагу перед попередніми технологіями у продуктивності, зберіганні повідомлень та ефективному розподілі. Забезпечує низьку затримку в передачі даних.

Таким чином Kafka ідеально підходить для створення додатків, які потребують більшої масштабованості та, відповідно, продуктивності.

2.2. Вибір мови програмування для реалізації серверної частини

Для вибору мов програмування та технологій, за допомогою яких буде написана серверна частина застосунку, розглянемо найпопулярніші мови програмування для розроблення backend-частини. На період квітня 2024, за даними сайту “Dou.ua” рейтинг найпопулярніших мов [11] для backend-розроблення очолюють Java – 14.7 %, Python – 13.0% і C# – 12.3%. Отже і обирати будем серед них.

2.2.1. Java

Почнемо з найбільш популярного на ринку рішення, з Java.

Java – об’єктно-орієнтована потужна та універсальна мова програмування. Універсальність цієї мови обумовлена тим, що скомпільований код на Java може запускатися на будь-якій платформі, для якої є реалізація віртуальної машини Java (JVM).

Розглянемо основні переваги:

- Кросплатформеність. Як вже згадувалось раніше, Java – застосунки виконуються на віртуальній машині JVM, яка реалізована для багатьох операційних систем.
- Універсальність. Завдяки наявності великої кількості різних бібліотек, фреймворків та технологій, мова підходить для виконання широкого спектру задач.
- Статична типізація. На відміну від Python, Java має статичну типізацію, що забезпечує значні переваги в порівнянні з мовами динамічного типу, а саме полегшене керування проєктами з великою кодовою базою та можливість оптимізувати середовище виконання.
- Швидкість та висока продуктивність. Хоча Java на JVM не може досягти такої ж продуктивності як компільовані мови такі як C++. Тим не менш удосконалення в роботі збирача сміття, використання динамічного компілятора JIT і безліч інших низькорівневих оптимізацій забезпечують високу продуктивність платформи Java.
- Об’єктно-орієнтованість. Користувачі у сучасному світі програмування сприймають це як належне, не помічаючи, що Java повністю об’єктно-орієнтована, і реалізує важливі концепції ООП, такі як: наслідування, інкапсуляція, поліморфізм, інтерфейси та композиція. Об’єктно-орієнтований дизайн є великою перевагою при проєктуванні складних застосунків, які орієнтовані на предметну галузь, де концепції реального світу повинні моделюватись в коді.
- Функціональний стиль. Зараз в індустрії розроблення програмного забезпечення все більше помічається тенденція в комбінації об’єктно-орієнтованого підходу з функціональним, оскільки функціональна частина допомагає зробити програми швидшими, менш багатослівними та легшими для розуміння. Java стала значною частиною цієї тенденції. З веденням 8 версії Java в неї був

запроваджений потужний фреймворк для функціонального обчислення Stream API, а також лямбда-вирази, що започаткувало еру функціонального програмування з Java.

- Зворотна сумісність. Оскільки Java просувається вперед з значними змінами в API, розробники надають перевагу зворотній сумісності. Навіть із запровадженням стандартних інтерфейсів і лямбда-виразів як функціонального програмування, Java не втратила свою зворотну сумісність. Код, написаний у попередніх версіях, без проблем може працювати в оновленому середовищі без необхідності повторної компіляції.
- Безпека. Безпека при використанні Java полягає в тому, що вона не використовує для об’єктів явні вказівники на область пам’яті, в якій ці об’єкти знаходяться, тобто програміст не може отримати доступ до пам'яті безпосередньо з коду. Крім того, програми написані на Java, запускаються всередині віртуальної машини JVM, для якої для динамічного завантаження класів надається завантажувач класів JRE, який відокремлює пакети класів локальної файлової системи від тих, які імпортуються ззовні.

Основними недоліками цієї мови програмування є:

- Продуктивність. Java потребує інтерпретації під час виконання. З одного боку, це дозволяє їй працювати в будь-якій операційній системі, але також робить її повільнішою, ніж такі мови, як C і C++.
- Споживання пам'яті. Оскільки виконувана програма Java працює на віртуальній машині JVM, системі необхідні додаткові ресурси пам'яті для забезпечення самої віртуальної машини.
- Багатослівність синтаксису. Програми написані на Java, як правило, вимагають більше рядків коду, ніж такі ж варіанти, написані на Python або Kotlin. Ця багатослівність призводить до довшого часу розроблення та більших зусиль на обслуговування.
- Обмеження низькорівневого доступу. Java навмисно обмежує

низькорівневий доступ до системних ресурсів з метою підвищення безпеки. Незважаючи на те, що це робить програми Java більш безпечними, це робить Java поганим варіантом для розроблення програмного забезпечення системного рівня, які вимагають апаратних маніпуляцій.

2.2.2. Python

Python – адаптивна, універсальна, високоефективна у використанні та розроблені інтерпретована мова програмування. Має підтримку різноманітних парадигм програмування та можливості автоматичного керування пам'яттю. Завдяки великій стандартній бібліотеці Python можна використовувати в багатьох сферах діяльності. Розглянемо Python більш детально, заглибившись в його переваги:

- Велика стандартна бібліотека. Стандартна бібліотека Python по справжньому величезна і в ній можна знайти майже всі необхідні для вирішення будь-якої задачі функції.
- Простий синтаксис. Не дивлячись на те, що Python – високорівнева, динамічно типізована мова програмування, вона має простий, схожий на англійську мову, синтаксис. Це полегшує читання та розуміння коду. Програмний код, написаний на Python, лаконічний в порівнянні з іншими мовами, такими як C/C++ і Java. Завдяки простоті синтаксису розробники можуть зосередитися на вирішенні бізнес задач і витрачати менше часу на розуміння синтаксису чи поведінки мови програмування.
- Інтерпретована мова. Основною відмінністю від основних компільованих мов, таких як C/C++ і Java є те, що в Python програмний код виконується послідовно, тобто без компіляції.
- Гнучкість. Велика кількість бібліотек та модулів від спільноти, які містять попередньо написаний програмний код, який легко можна інтегрувати в існуючу систему. А також можливість інтегрувати

Python для використання з іншими мовами програмування, такими як C++ і Java. Це дає можливість використовувати сильні сторони Python в контексті певної функціональності та сценаріїв інших мов програмування.

- Висока масштабованість. Python легко використовувати для масштабування будь-яких процесів обчислень. Це можуть бути як розподіленні обчислення, так і обробка і аналіз великих наборів даних. Це робить Python популярним вибором для розподілених обчислень і машинного навчання.
- Динамічна типізація. При невеликому обсягу кодової бази динамічна типізація полегшує процес розроблення та тестування програмного забезпечення, оскільки програмісту не потрібно турбуватися про тип даних оголошених змінних.
- Швидкість розроблення. Python простий за синтаксисом та у використанні. Він не потребує процесу компіляції, тому розроблення програм потребує менше часу.
- Портативність. Інтерпретатор запускає справжній файл с Python кодом, а не його компільовану версію, це означає, що програмне забезпечення, написане на Python, може працювати на будь-якій системі, для якої є підтримка інтерпретатора Python.

Розглянемо недоліки:

- Низька ефективність використання пам'яті. Динамічна типізація та інтерпретоване виконання закономірно призводять до більшого використання пам'яті в порівнянні з компільованими мовами програмування. Процес збирання сміття в Python є додатковим недоліком, оскільки збирач сміття може не завжди швидко та ефективно звільняти пам'ять. Це може призводити до витоків пам'яті та додаткових проблем пов’язаних з цим. Для програм написаних на Python потрібно ретельне планування використання ресурсів та оптимізація процесу їх звільнення. Всі перераховані фактори

роблять Python не найкращим рішенням для додатків, що працюють довго на середовищах з обмеженою пам'яттю.

- Повільна швидкість. Одним з основних недоліків Python в порівнянні з компільованими мовами є те, що він повільніший. Знову ж таки, інтерпретація призводить до рядкового виконання коду і, відповідно, до зменшення швидкості. Динамічна типізація додатково сповільнює процес виконання коду, оскільки інтерпретатору додатково доводиться виконувати роботу по аналізу типу даних змінних.
- Неоптимізований доступ до бази даних. На відміну від Java та інтерфейсу Java Database Connectivity (JDBC), Python не має потужного, високоякісного та легкого у використанні інструменту для роботи з базами даних. Це робить Python менш привабливим в контексті написання великих програмних застосунків для складної взаємодії з базою даних.
- Помилки виконання. Безсумнівно, динамічна типізація додає зручності в процесі створення програмного забезпечення, проте також додає додаткових проблем пов'язаних з помилками виконання. Тип даних будь-якої змінної під час процесу виконання може змінитися. Помилки пов’язані з цим могли б бути виявленні на етапі компіляції, але Python – інтерпретована мова. Це робить помилки виконання складно прогнозованими та додає необхідності ретельно тестувати програмне забезпечення.
- Погане рішення для великого проєкту. Відсутність суворої типізації ускладнює процес розроблення, підтримки і налагодження коду для великих проєктів. Це призводить до збільшення часу і витрат на розробку, а також до потенційних помилок і вразливостей системи.
- Погана багатопотоковість. Не дивлячись на те, що Python має модуль багатопотоковості, не можна його назвати багатопотоковим через глобальне блокування інтерпретатора (GIL). Це блокування

гарантує те, що тільки один потік виконання може виконувати байт-код Python. Навіть у багатоядерних системах декілька потоків виконання не можуть виконуватись паралельно.

2.2.3. C#

C# – створена компанією Microsoft об'єктно-орієнтована мова програмування для створення застосунків на базі платформи .NET [15]. Потужна і універсальна мова програмування, яка для створення багатофункціональних і високоякісних застосунків пропонує розробникам використовувати широкий спектр бібліотек і інструментів .NET. Серед переваг цієї мови програмування можна виділити:

- Високорівнева мова з можливістю доступу до пам'яті. Однією з важливих особливостей середньо- та низькорівневих мов програмування є те, що вони можуть напряму звертатися до пам'яті, використовуючи спеціальні типи команд, які називаються вказівниками. Не дивлячись на те, що C# вважається високорівневою мовою програмування, розробниками, як і раніше, доступна обмежена функціональність роботи з вказівниками.
- Кросплатформеність. Принцип виконання програмного коду застосунку написаного мовою C# дуже схожий с принципом запуску Java коду на віртуальній машині JVM. Реалізація віртуальної машини для C# називається Common Language Runtime (CLR).
- Типобезпека з динамічними можливостями. C# є типобезпечним, тобто зміна не може змінити свій тип під час виконання. Перевіряється це на етапі компіляції. В такому випадку вдається відловити помилки типізації до того, як вони потраплять в середовище виконання. Проте, варто зазначити, що мова підтримує також динамічну типізацію, тобто є можливість створювати об'єкти і помічати їх динамічним типом. В такому випадку, помилка типізації буде виявлятись на етапі виконання.

- Сумісність с іншими .NET мовами. Це є однією з особливостей C# і платформи .NET в цілому. Код написаний на C# може взаємодіяти з застосунками написаними на сумісних мовах, таких як F#, C++, Visual Basic, Windows PowerShell.

Серед недоліків можна виділити:

- Залежність від платформи .NET. Значною мірою C# залежить від ресурсів .NET для роботи на різних операційних системах. Якщо не розглядати .NET як основний технологічний стек, то C# сам по собі не є таким гнучким. Якщо брати в порівнянні, програма, скомпільована на Java, буде працювати на будь-якій платформі для якої є реалізація JVM, в той час як для виконання C# необхідно використовувати різні середовища виконання для різних платформ, відповідно адаптувати код до системних вимог.
- Витрата ресурсів. Як вже раніше згадувалось, середовище виконання для програмного забезпечення, написаного на C#, - це віртуальна машина. Це вимагає додаткових ресурсів і робить застосунки, написані на C#, менш придатними для створення програм для малопотужних пристроїв.
- Продуктивність. Якщо порівнювати з найближчим аналогом Java, C# з точки зору продуктивності має майже ідентичні показники, але якщо порівнювати с C++, то існує суттєва різниця в продуктивності, в якій C++ сильно випереджає C#.

Враховуючи вищезазначені особливості, переваги та недоліки досліджуваних мов програмування, побудуємо порівняльну таблицю. Ця таблиця дозволить краще зрозуміти, які мови програмування можуть бути найбільш ефективними для реалізації серверної частини. Важливо враховувати такі аспекти, як кросплатформеність, підтримка декількох парадигм програмування, швидкодія, простота використання, безпека, масштабованість, робота з базами даних, сумісність з іншими мовами програмування, універсальність та модульність.

Таблиця 2.1

Порівняння можливостей мов програмування для реалізації серверної частини

Критерій порівняння \ Мова програмування	Java	Python	C#
Кросплатформеність	+	+/–	+/–
Підтримка декількох парадигм програмування	+	+	+
Висока швидкодія	+/–	–	+/–
Простота використання	+/–	+	+/–
Ефективне використання ресурсів	+	–	+
Безпека	+	–	+/–
Підходить для створення великого проєкту	+	–	+
Масштабованість	+	–	+
Підтримка бібліотек та фреймворків	+	+	+
Розвинута спільнота та екосистема	+	+	+
Здатність інтеграції з різноманітними технологіями	+	+	+
Можливість ефективно працювати з базою даних	+	–	+
Підтримка багатопотоковості	+	–	+
Сумісність з іншими мовами програмування	+	+	+
Універсальність	+	+	+
Модульність	+	+	+

Отже, враховуючи вищезазначені факти, для розроблення серверної частини було обрано Java, як основну мову програмування.

2.3. Технології для реалізації серверної частини

В даному пункті дипломного проєкту для обраної мови програмування будуть розглянуті найпопулярніші фреймворки, бібліотеки і технології для імплементації серверної частини застосунку.

В якості основної платформи для створення серверної частини було обрано Spring Framework [16].

Spring Framework – надійна і найбільш популярна платформа для створення Java застосунків. Spring можна по-справжньому назвати найкращим Java-фреймворком для веброзроблення. Продукти, розроблені з використання Spring, мають високі показники безпеки та швидкості. По своїй суті Spring Framework – контейнер для ін'єкції залежностей. Розбиття застосунку на логічні шари, такі як: доступу до бази даних, проксі, АОП, RPC, MVC дозволяє швидко та зручно створювати застосунки на Java. Ключовим в Spring є висока інфраструктурна підтримка. Саме через це Spring зарекомендував себе як ідеальне рішення для створення корпоративних застосунків.

Не дивлячись на те, що де-юре Spring називається фреймворком, де-факто це назва для широкого кола фреймворків, кожен з яких сфокусований на вирішенні певних задач. Об'єднані ці фреймворки загальною модульною структурою. Це дозволяє швидко обирати і підключати тільки необхідні для застосунку компоненти. Серед найпопулярніших компонентів екосистеми Spring можна виділити:

- Spring Data – компонент, призначений для роботи з реляційними та нереляційними базами даних. Складається з різноманітних інтерфейсів та інструментів для роботи з даними через JPA.
- Spring Security – компонент, призначений для роботи з авторизацією та аутентифікацією.

- Spring Web – компонент для роботи з мережевими даними та іншими функціональними модулями, такими як: браузер, frontend та мобільні застосунки, мікросервіси.
- Spring AOP – компонент для інтеграції аспектно-орієнтованого програмування.
- Spring Boot – компонент-додаток до Spring, призначений для полегшення і прискорення роботи. Надає набір утиліт для автоматизації налаштувань фреймворку. Автоматично конфігурує збірку додатку за допомогою зовнішніх залежностей. Створює та запускає веб-сервер/контейнер сервлетів.

Отже, підсумуємо основні переваги та недоліки використання Spring Framework.

Переваги:

- Гнучкість. Spring надає доступ до великої кількості бібліотек, модулів та компонентів. Розробник може змінювати конфігураційні параметри застосунку за допомогою XML або Java анотацій. Функціональність ін’єкції залежностей забезпечує широкий спектр можливостей.
- Абстракція. Використання специфікацій JMS, JDBC, JPA та JTA забезпечує потужну абстракцію.
- Життєвий цикл. Spring бере на себе велику частину відповідальності управління життєвим циклом компонентів.
- Спрощення тестування. Використання ін'єкції залежностей суттєво спрощує процес тестування.
- Безпека. Регулярні оновлення системи безпеки і ретельне відслідковування сторонніх залежностей забезпечують безпеку даних і застосунків.
- Підтримка. Високий рівень підтримки зі сторони спільноти та розробників фреймворку а також велика кількість посібників.

- Слабка зв’язність. Механізм ін'єкції залежностей забезпечує слабку зв'язність компонентів системи.
- Вбудований сервер. Компонент Spring Boot містить вбудовані контейнери сервлетів, такі як Tomcat і Jetty. Це полегшує розробникам збірку проєкту в JAR-пакет та розгортання його в середовищі виконання.
- Мікросервіси. Підтримка мікросервісної архітектури, а саме наявність компонента Spring Cloud, необхідного для створення та керування розподіленими системами.

Недоліки:

- Складність. Використання фреймворку Spring потребує досвіду розроблення. Також деякі компоненти можуть бути складними для використання та опанування.
- Накладні витрати. Spring є не найкращим рішенням для простих проєктів, оскільки автоматична конфігурації Spring Boot може призвести до додаткових витрат пам'яті, на відміну від облегшених фреймворків.
- Розростання залежностей. Автоматичне під’єднання залежностей може також призвести до під’єднання непотрібних бібліотек.

Як інструмент об'єктно-реляційного відображення (ORM) в контексті Spring Boot застосунку будемо використовувати Hibernate.

Об'єктно-реляційне відображення необхідне для інтуїтивної роботи з базою даних, використовуючи об'єкти Java замість написання складних SQL запитів.

Hibernate має можливість працювати з будь-яким типом баз даних, як реляційних так і нереляційних. Це забезпечує гнучкість системи, забезпечує захист і дозволяє розробникам змінювати систему БД без переписування коду. Використання кешу першого рівня для збереження даних, до яких часто звертаються, та ліниве завантаження, тобто відкладення завантаження

об'єктів до їх безпосереднього використання, значно підвищує швидкість та ефективність застосунків.

Ще однією значною проблемою, яку вирішує Hibernate, є узгодженість та цілісність даних. Він дозволяє визначити межі транзакцій та в автоматичному режимі виконувати відкати та підтвердження операцій, що мінімізує можливість пошкодження даних.

Клієнтом для забезпечення комунікації між сервісами було обрано Spring Cloud OpenFeign.

Spring Cloud OpenFeign – реалізація декларативного HTTP клієнта для програм на Spring Boot. Сервіс Feign був розроблений компанією Netflix для створення HTTP клієнтів за допомогою анотованих інтерфейсів без використання шаблонного коду. Реалізація Spring Cloud OpenFeign зручно інтегрується між хмарними сервісами інфраструктури Spring і забезпечує балансування навантаження між сервісами.

Для контролю версій баз даних було порівняно два таких продукти як Liquibase та FlywayDB.

Обидва цих інструментів реалізують принцип еволюції баз даних. Як і FlywayDB, так і Liquibase написані на Java, що полегшує їх інтеграцію до Spring Boot застосунка. Розглянемо основні функціональні можливості які пропонують ці продукти:

- контроль версій баз даних;
- аудит змін моделей баз даних;
- наявність таблиць з контрольною сумою;
- інкрементне розгортання скриптів;
- запобігання розгортання застосунку при десинхронізації з базою даних.

Як видно, обидва ці інструменти пропонують всі необхідні функції для автоматизації процесу рефакторингу і еволюційної моделі баз даних.

В якості системи міграцій бази даних було обрано Liquibase, оскільки вона має більш широку та гнучкішу функціональність, а саме:

- можливість вказувати зміни в різних форматах, таких як: SQL, XML, YAML, JSON;
- міграція Liquibase не потребує дотримання правил іменування файлів;
- можливість налаштовувати порядок виконання змін;
- можливість робити знімок поточного стану бази даних з метою порівняння з іншою БД.

Оскільки планується впровадження мікросервісної архітектури для розроблюваного застосунку, постає питання стосовно моніторингу та відлову помилок в системі. У зв’язку з цим планується інтегрувати набір компонентів для забезпечення централізованого збереження логів з різних сервісів. Назва цього стеку – це абревіатури перших трьох літер відкритих проєктів: **E**lasticsearch, **L**ogstash, **K**ibana (ELK). Розглянемо більш детально кожен з цих проєктів:

Elasticsearch – розподілений пошуково аналітичний двигун. Розроблений на основі системи повнотекстового пошуку Apache Lucene. Основне призначення – це можливість швидкої та ефективної роботи з великими обсягами структурованих і неструктурованих даних. Повнотекстовий пошук дозволяє знаходити відповідності між записами та документами, навіть якщо вони мають велику кількість даних. Також присутня можливість проводити аналіз даних, агрегацію, фільтрацію, групування та обчислення. В контексті збору логів Elasticsearch може індексувати логи по мірі надходження та виконувати запити до них у режимі реального часу.

Logstash – конвейер з оброблення даних одночасно для декількох джерел введення. Передбачає декілька етапів оброблення вхідних даних:

- Input – перетворення логів в зрозумілий для комп’ютера вигляд.
- Filter – процес структурування, нормалізації та фільтрації логів

відповідно до набору умов для виконання певної дії чи події.

- Output – фінальний процес для перетворення логів в формат в якому вони будуть надсилатися до системи збереження, включаючи Elasticsearch або Kafka.

Kibana – веб інструмент для візуалізації та аналізу даних. Дозволяє візуалізувати індексовані дані в системі Elasticsearch. Представлення, створене Kibana може набувати формату графіку, діаграми, таблиці або списку. Також є можливість фільтрувати дані за різними параметрами. Використовуючи гнучкі налаштування можна відслідковувати шлях по якому відбувається виклик певних функцій застосунку.

Отже, інтеграція ELK стеку дозволить швидко і безпечно отримувати та аналізувати системні логи в режимі реального часу. Це допоможе виявляти будь-які проблеми пов'язані з серверною частиною застосунку.

2.4. Технологічний стек для реалізації клієнтської частини

Найкращим рішення для забезпечення максимальної кросплатформеності є створення вебзастосунку. На більшості сучасних персональних мобільних пристроях попередньо встановлений або доступний для встановлення веббраузер. На відміну від спеціалізованих програмних застосунків, доступних лише на певних операційних системах, браузер дозволяє користувачам отримати доступ до ресурсів з будь-якого пристрою, включаючи комп’ютери, планшети і смартфони.

Одними з найпоширеніших технологій для розроблення клієнтської частини вебзастосунку є Angular, Vue і React. Порівняємо їх:

2.4.1. React

В загальному плані, краще назвати React бібліотекою для суттєвого спрощення побудови та менеджменту DOM елементів. Назвати його фреймворком складно, оскільки він не містить ні Routing, ні інших додаткових можливостей.

Компоненти в React мають вигляд функцій, які приймають параметри та повертають щось схоже HTML розмітку. React використовує спеціальний синтаксис JSX (JavaScript Extended). Цей синтаксис – «синтаксичний цукор» для мови JavaScript, який дозволяє поєднувати HTML і JavaScript.

2.4.2. Angular

Angular – повноцінний, керований MVVM принципами побудови застосунків фреймворк від компанії Google, оснований на використанні модульної, компонентної і сервісної структури. Важливою особливістю Angular є те, що він створювався для розроблення великих застосунків і, на відміну від Vue і React, які використовуються для проєктів будь-якої складності, для Angular є підтримка використання мови TypeScript без встановлення будь-яких програмних розширень.

Використання статичнотипізованого TypeScript для написання Angular вебзастосунків Angular є найкращим вибором для інтеграції з серверною частиною, написаною на C#, C++, Java.

Розбиття клієнтської частини на модулі дозволяє відокремлювати певні частини коду, призначеного для домену, відокремлювати компоненти, сервіс-провайдера та інші частини застосунку.

В порівнянні з React, Angular має дещо складнішу структуру. Окрім компонентів, в Angular з’являються додаткові елементи такі як:

- пайпи – функції для перетворення візуального вигляду даних HTML темплейту, що дозволяють зручно форматувати числа, дати, валюти та інше;
- компоненти – елемент для візуального відображення даних, що поєднує HTML розмітку, бізнес логіку та стилі;
- директиви – класи для забезпечення додаткових можливостей для користувацьких елементів розмітки та іншим елементам застосунку;
- сервіси – класи для централізованого зберігання даних, надання функціональних та утилітарних методів для роботи з зовнішніми

сервісами та репозиторіями, що додаються за допомогою механізму ін'єкції залежностей;

- модулі – частина абстрагування для групування всіх раніше зазначених елементів.

2.4.3. Vue

Vue – найновіший серед розглянутих фреймворк. Фактично є сумішшю Angular та React. З одного боку, це просто інструмент для роботи з DOM, з іншого – цілий набір засобів, що має власний Vue CLI та розгалужену екосистему бібліотек. Основна філософія Vue полягає в підході написання компонентів в одному файлі. Тобто стилі, розмітка та бізнес логіка компоненту має знаходитися в одному файлі.

Таблиця 2.2

Порівняння можливостей технологій для реалізації клієнтської частини

Критерій порівняння / Технологія	React	Angular	Vue
Стабільність та підтримка технології	+	+	+/–
Наявність детальної документації	+	+	+/–
Наявність екосистеми допоміжних бібліотек	+	+	+
Підтримка TypeScript	+	+	–
Найкраще рішення для великого проєкту	–	+	–
Легкість оновлення версії	+	+	–
Регулярне оновлення та вдосконалення	+	+	–

Отже, з огляду на вище описані пункти та загальний технологічний стек, найкращою комбінацією для написання клієнтської частини буде використання Angular в якості основного фреймворку та TypeScript в якості мови програмування.

2.5. Вибір СУБД

При виборі системи управління базою даних і бази даних в цілому слід відштовхуватись від того, що використання мікросервісної архітектури дозволяє не зав’язуватись на використанні лише однієї СУБД та БД. Найкращим рішенням буде використання різних типів БД для вирішення конкретних задач.

Багато компаній використовують комбінацію реляційних та нереляційних баз даних. NoSQL бази даних виділяються своєю швидкодією та масштабуванням, проте, в деяких випадках краще надати перевагу реляційному підходу до збереження даних.

Зазвичай при ситуації, де необхідною є відповідність ключовим вимогам до баз даних ACID (Atomicity, Consistency, Isolation, Durability – атомарність, консистентність, ізольованість та довговічність), найкращим рішення буде використати реляційну базу даних. Це забезпечить низьку ймовірність виникнення неочікуваної поведінки системи та забезпечить цілісність даних. Досягається це шляхом виконанням правил транзакційної взаємодії з базою даних, що суттєво відрізняє реляційні бази даних від нереляційних.

Ще одним пунктом для вибору реляційної бази даних може бути вимога до жорсткої структурованості, що гарантує запобігання випадкових змін.

Для нереляційних баз даних притаманне горизонтальне масштабування, і це краще для роботи з великими обсягами даних.

Основною перевагою при використанні NoSQL баз даних є можливість зберігання різноманітних атрибутів та полів, не пов’язуючись

до жорстких обмежень стосовно схеми. Це робить нереляційні сховища ідеальним рішенням для неструктурованих або недостатньо структурованих даних.

Отже, як вже зазначено попередньо, для розроблення застосунку було обрано комбінацію реляційної та нереляційної бази даних.

В якості СУБД для реляційної бази даних було обрано MySQL через високу продуктивність і можливість оброблювати великі об’єми даних. Вона оптимізована до інтенсивних робочих навантажень та має швидку систему індексування.

Для нереляційного збереження даних було обрано документно-орієнтовану базу даних MongoDB.

Також для проведення інтеграційного та E2E тестування, передбачено використання вбудованої SQL бази даних H2.

2.6. Висновки до розділу 2

На основні порівняльного аналізу, проведеного в рамках даного дипломного проєкту, розглянуто мови та технології для реалізації серверної частини застосунку. Обрано мову програмування сімейства JVM, а саме Java. В якості основного фреймворку для використання при backend розроблені обрано Spring Framework.

По широкому набору характеристик технологічних стеків та оцінці їх переваг та недоліків, для створення клієнтської частини вебзастосунку було обрано мову програмування TypeScript і відповідний фреймворк Angular.

Для забезпечення максимально ефективної обробки даних в межах розроблюваного застосунку було розглянуто декілька можливих видів СУБД і БД, та обрано комбінацію з реляційних MySQL і H2, та нереляційної MongoDB.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ ЗАСТОСУНКУ

3.1. Вимоги до розроблюваного вебзастосунку

Детальна підготовка і документування вимог необхідні для будь-якого проєкту з розроблення програмного забезпечення. Однією з причин невдачі є погане визначення вимог на старті розроблення. Існує два основні типи вимог: функціональні та нефункціональні.

Функціональними називають вимоги що визначають особливості та функції системи [26]. Ці вимоги описують, що саме повинне виконувати програмне забезпечення за нормальних умов для забезпечення потреб користувачів. Для розробника це набір можливостей, які необхідно реалізувати. Це можуть бути як і основні функції застосунку, так і допоміжний спрямовані на покращення взаємодії з користувачем, який є не обов’язково необхідним для функціонування системи.

Нефункціональні вимоги – це набір специфікацій, які описують обмеження або експлуатаційні можливості системи. Також це опис критеріїв, що використовуються для оцінки роботи системи і не належать до поведінки чи функціональності.

3.1.1. Аналіз проблематики та цілей вебзастосунку

Успішна реалізація вебзастосунку потребує попереднього аналізу проблематики та цілей. Це необхідно для задоволення потреб користувачів та досягнення поставлених задач.

Першочерговою задачею при розробленні вебзастосунку є аналіз проблематики вебзастосунку. Цей аналіз полягає в дослідженні контексту використання та виявленні основних проблем та потреб, які застосунок має вирішувати. Це ключовий етап, оскільки саме він дозволяє визначати реальні потреби користувачів та забезпечує відповідність кінцевого продукту цим потребам.

Відповідно до обраної предметної галузі не складно створити портрет потенційного користувача розроблюваного застосунку.

Потенційний користувач, переважно чоловік від 20 до 50 років, зацікавлений у використанні застосунку для відстеження та ведення обліку технічного стану транспортного засобу.

При виборі застосунку для користування користувач, в першу чергу, звертатиме увагу на кількість функціональних можливостей та на простоту і лаконічність дизайну.

Основними проблемами користувача будуть:

- паперові документи, чеки, гарантії (паперові документи можуть бути втрачені або пошкоджені через різні чинники, такі як волога, вогонь або інші фактори);
- відсутність або не наявність при собі сервісної книжки автомобіля;
- прогнозування майбутніх та ведення статистики поточних витрат (ручне ведення статистики витрат може бути часомістким і вимагати значних зусиль, а без ведення статистики поточних витрат прогнозування майбутніх є неможливим);
- забування про технічне обслуговування (автовласники забувають про необхідність проведення регулярного технічного обслуговування);
- складнощі в передачі повної інформації про стан автомобіля та історії обслуговування наступним власникам;
- недовіра потенційних покупців (потенційні покупці можуть відчувати недовіру до факту проведення технічних процедур, що ускладнює процес продажу автомобіля і знижує його ціну).

На основі перелічених проблем користувача, побудуємо дерево проблем, що зображене на рис. 3.1.

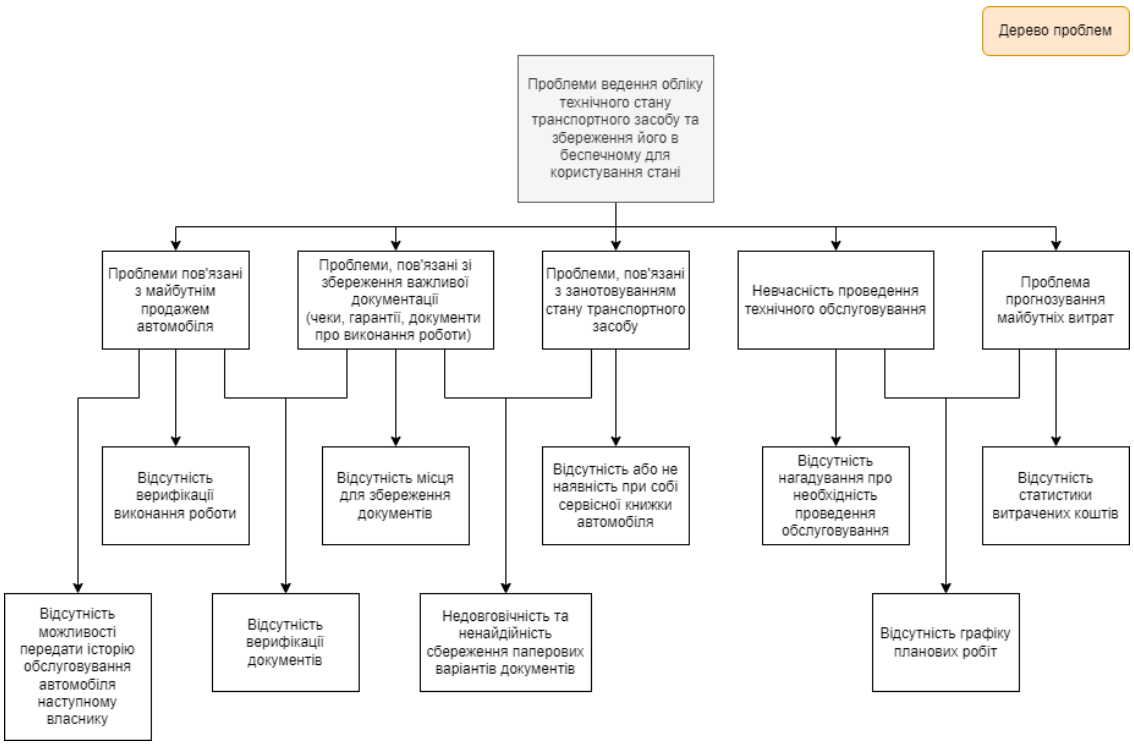


Рис. 3.1. Дерево проблем користувача

Наступним кроком після аналізу проблематики є визначення цілей, які повинні бути досягнені, за допомогою використання вебзастосунку. Враховуючи вищезазначені проблеми користувача, можна визначити цілі для вебзастосунку. Цілі, об’єднанні ієрархічною структурою, продемонстровані на рис. 3.2.

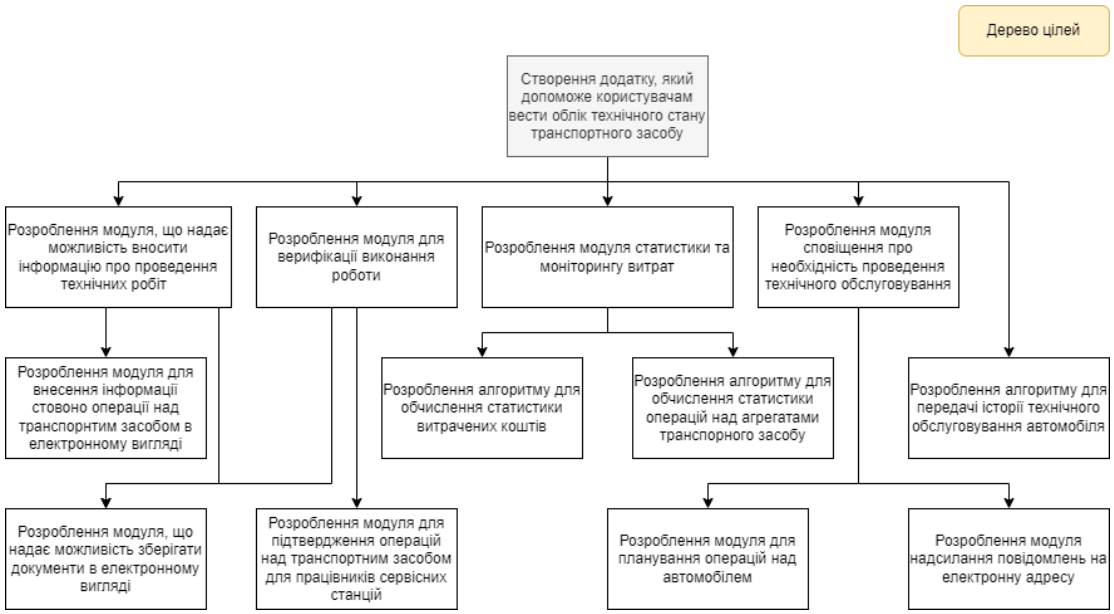


Рис. 3.2. Дерево цілей розроблення вебзастосунку

3.1.2. Функціональні та нефункціональні вимоги

Визначимо основні функціональні вимоги для різних категорій користувачів, які система повинна забезпечувати. Перелік функціональних вимог наведено у табл. 3.1-3.3.

Таблиця 3.1

Функціональні вимоги до роботи застосунку для неавторизованих користувачів та неавторизованих працівників СТО

Код вимоги	Опис вимоги
F-1	Можливість реєстрації звичайного користувача у системі. Для реєстрації користувач має вказати такі дані: • електронну адресу; • пароль; • ім’я; • прізвище; • адресу; • дату народження; • бажану мову системи. Поле електронної адреси має бути заповнене адресою відповідно шаблону, інакше користувач має бути попереджений про неправильність формату електронної адреси та заблокований від завершення процесу реєстрації. Якщо будь-яке поле форми реєстрації залишається незаповненим, користувач також має бути попереджений та заблокований від завершення процесу реєстрації.
F-2	Можливість авторизації для неавторизованого користувача або неавторизованого працівника СТО за допомогою електронної пошти та паролю. У разі вводу некоректних даних, відсутності зареєстрованого користувача або сервісної станції в системі, або неправильності формату електронної адреси, система має попередити про це та заблокувати від завершення процесу авторизації.

Таблиця 3.2

Функціональні вимоги до роботи застосунку з авторизованими користувачами

Код вимоги	Опис вимоги
F-3	Можливість користувачу переглядати сторінку власного профілю.
F-4	Користувач повинен мати можливість модифікувати персональну інформацію. Вікно модифікації персональної інформації повинне містити поля: • електронну адресу; • ім’я; • прізвище; • адресу; • дату народження. Поле електронної адреси повинно бути заблокованим для модифікації. У разі якщо будь-яке поле форми залишається незаповненим, користувач має отримати попередження від системи про це, та має бути заблокований від завершення процесу модифікації даних.
F-5	Користувач повинен мати можливість зміни пароль персонального аккаунта. Вікно модифікації повинно мати поле для введення нового паролю та поле підтвердження введеного паролю У разі порушення новим паролем безпековим вимогам або невідповідності повторно введеного паролю користувач має бути попереджений системою та заблокований від завершення процесу зміни паролю.

Продовження табл. 3.2

F-6	Можливість вийти з системи.
F-7	Можливість змінити мову інтерфейсу на англійську або українську. При зміні мови та повторному вході, система має запам’ятовувати попередній вибір мови
F-8	Можливість переглянути всі транспортні засоби користувача і інформацію стосовно них. Наявність опції сортування транспортних засобів за пробігом, назвою або кількістю обслуговувань.
F-9	Користувач повинен мати можливість додати новий транспортний засів. Вікно додавання транспортного засобу повинне містити форму з такими полями та елементами: • форма вибору зображення автомобіля; • марка; • модель; • рік виготовлення; • ідентифікаційний номер транспортного засобу; • реєстраційний номер; • пробіг; • тип двигуна; • об’єм двигуна; • тип палива; • тип трансмісії; • колір; • тип кузова. У разі вводу некоректних даних (від'ємний пробіг або об'єм двигуна), а також у випадку не повного заповнення форми, система має попередити користувача та заблокувати його від створення транспортного засобу.

Продовження табл. 3.2

F-10	Користувач повинен мати можливість передати історію технічного обслуговування наступному власнику. Вікно передачі власності повинно мати поле для електронної адреси майбутнього власника. У разі не заповнення цього поля, некоректного формату електронної адреси або відсутності попередньо зареєстрованого користувача з такою адресою, система має попередити про це та заблокувати завершення процесу передачі історії обслуговування.
F-11	Користувач повинен мати можливість виконати видалення транспортного засобу, яке потребує підтвердження.
F-12	Можливість переглянути статистику грошових витрат та затраченого часу. Статистика витрат та затраченого часу має бути представлена в форматі секторної діаграми. При наведенні на певний сектор мають демонструватися затрати для відповідного транспортного засобу.
F-13	Можливість переглянути графік кількості щомісячних витрат на обслуговування всіх або обраного транспортного засобу. Графік має бути представлений в форматі лінійної діаграми. При наведенні на точки діаграми мають відображатися затрати на певний період. Також на лінійній діаграмі має бути присутня лінія тренду.
F-14	Можливість перегляду статистики операцій за певним вузлом всіх або обраного транспортного засобу. Статистика операцій має бути представлена в форматі гістограми, де категорія гістограми представлена вузлом, а абсолютне значення – кількість операцій, виконаних над ним.
F-15	Користувач повинен мати можливість переглядати список операцій та інформацію, що стосується цих операцій, над транспортним засобом, згрупованих за категоріями. Наявність опції сортування записів за пробігом, ціною, датою та верифікацією.

Продовження табл. 3.2

F-16	Наявність особливої відмітки у верифікованого запису та назви СТО, в якій проводилося технічне обслуговування.
F-17	Користувач повинен мати можливість додавати записи про проведення технічного обслуговування. Вікно додавання запису повинно містити форму з такими полями та елементами: • випадаючий список для вибору категорії обслуговування; • назва обслуговування; • деталі; • пробіг; • затрачений час; • ціна; • дата наступного обслуговування (не обов'язково); • список запчастин та матеріалів, використаних під час обслуговування (не обов'язково); • список документів про виконання роботи (не обов'язково). Після додавання запчастини, матеріалу чи документу біля них повинна з’явитися кнопка для вилучення їх зі списку. Вилучення повинне супроводжуватися підтвердженням операції. У разі некоректно введених даних (від'ємний пробіг чи ціна), а також у випадку не заповнення обов'язкових елементів форми, система повинна попередити користувача та заблокувати його від створення запису. Якщо вказана дата наступного обслуговування, система повинна записати цю інформацію в графік запланованих подій та у відповідну дату надіслати повідомлення про необхідність проведення технічного обслуговування.
F-18	Користувач повинен мати можливість видаляти записи про технічне обслуговування, якщо вони неверифіковані та створені саме ним. Видалення повинне супроводжуватися підтвердженням операції.

Продовження табл. 3.2

F-19	Користувач повинен мати можливість завантажити документи закріплені за певним записом.
F-20	Можливість переглядати графік запланованих робіт, пов'язаних з технічним обслуговуванням, в форматі календаря. Можливість переглядати заплановані роботи за місяцями.
F-21	Надсилання системою повідомлення на електронну пошту про необхідність проведення обслуговування відповідно графіка запланованих робіт.
F-22	Можливість додавання події до графіка запланованих робіт. Вікно додавання події повинне містити форму з такими полями та елементами: • випадаючий список для вибору автомобіля стосовно якого ця поді; • випадаючий список для вибору категорії обслуговування; • назва обслуговування; • дата нагадування; • пробіг. У разі некоректно введених даних, а також у випадку не заповнення обов'язкових елементів форми, система повинна попередити користувача та заблокувати його від створення події.
F-23	Можливість передчасного виконання події, включеної до графіка запланованих робіт, що призводить до скасування повідомлення. Виконана подія повинна відображатися в календарі як виконана.
F-24	Можливість видалення події з графіка запланованих робіт. Видалення повинне супроводжуватися підтвердженням операції.

Таблиця 3.3

Функціональні вимоги до роботи застосунку з авторизованими працівниками СТО

Код вимоги	Опис вимоги
F-25	Працівник СТО повинен мати можливість пошуку автомобілів користувача по електронній пошті. Поле вводу електронної пошти повинне бути заповнене адресою відповідно шаблону, інакше система має попередити працівника СТО про неправильність формату електронної адреси. Для знайдених автомобілів має бути передбачена функціональність сортування.
F-26	Працівник сервісної станції повинен мати можливість переглядати список операцій та інформацію, що стосується цих операцій, над транспортним засобом, згрупованих за категоріями. Для записів має бути передбачений функціонал сортування записів за пробігом, ціною, датою та верифікацією.
F-27	За аналогією до функціональної вимоги F-17, працівник СТО повинен мати можливість додавати записи про проведення технічного обслуговування. Записи, додані працівником СТО, повинні відображатися користувачеві як верифіковані.
F-28	Можливість видаляти записи про технічне обслуговування, якщо вони створені саме цією сервісною станцією. Видалення повинне супроводжуватися підтвердженням операції.
F-29	Можливість завантажити документи закріплені за певним записом.

Після формулювання функціональних вимог наступним логічним кроком є визначення нефункціональних вимог. Нефункціональні вимоги не визначають, що система робить, але ставлять умови щодо якості, надійності, продуктивності, безпеки та інших аспектів, які впливають на взаємодію з

системою. Нефункціональні вимоги описують широкий спектр характеристик системи, включаючи якість виконання, надійність її функціонування протягом тривалого часу, продуктивність (швидкість реакції та обробки даних), узгодженість даних, зручність використання (легкість навчання та зручність для користувача), масштабованість (здатність системи адаптуватися до зростаючих потреб), та підтримку та обслуговування. Перелік нефункціональних вимогу описаний нижче у табл. 3.4-3.8.

Таблиця 3.4

Нефункціональні вимоги до безпеки застосунку

Код вимоги	Опис вимоги
SEC-1	Паролі користувачів повинні зберігатися в захешованому форматі.
SEC-2	Система повинна мати механізми контролю доступу, які дозволяють обмежувати доступ до конфіденційної інформації тільки для авторизованих користувачів з відповідними правами.
SEC-3	Система повинна використовувати механізми, такі як токени JWT, для запобігання перехоплення сесій.

Таблиця 3.5

Нефункціональні вимоги до продуктивності застосунку

Код вимоги	Опис вимоги
PER-1	Система повинна забезпечувати час відповіді на запити користувачів менше ніж 3 секунди у 95% випадків.
PER-2	Система повинна ефективно використовувати ресурси сервера та мережі для забезпечення оптимальної продуктивності.

Таблиця 3.6

Нефункціональні вимоги до масштабованості застосунку

Код вимоги	Опис вимоги
SCA-1	Архітектура системи повинна бути спроектована з урахуванням можливості масштабування горизонтально та вертикально для збільшення пропускної здатності при збільшенні обсягу користувацького трафіку.
SCA-2	База даних повинна бути спроектована з урахуванням потенційного зростаємого обсягу даних та навантаження.

Таблиця 3.7

Нефункціональні вимоги до сумісності застосунку

Код вимоги	Опис вимоги
COM-1	Клієнтська частина вебзастосунку повинна бути сумісною з популярними веб-браузерами, такими як Chrome, Opera, Firefox, Safari та Edge.
COM-2	Інтерфейс вебзастосунку повинен мати адаптивний дизайн, що забезпечує правильне відображення на різних типах пристроїв, таких як мобільні пристрої, планшети та комп’ютери.

Таблиця 3.8

Нефункціональні вимоги до надійності застосунку

Код вимоги	Опис вимоги
REL-1	Система за будь-яких умов повинна забезпечувати цілісність даних.
REL-2	Система повинна мати механізми моніторингу для виявлення та вирішення відмов у реальному часі.

3.2. Структурна організація застосунку

Обраний для розроблення підхід з використання клієнт-серверної архітектури передбачає розділення функціональності програмного забезпечення відповідно на дві складові: клієнтську та серверну.

Клієнтська частина є тим, що безпосередньо взаємодіє з користувачем. Фактично, це інтерфейс, доступний користувачам для взаємодії з системою через браузер. Саме клієнтська частина відповідальна за візуальне представлення інформації та обробку користувацьких дій. В якості фреймворку для розроблення клієнтської частини вебзастосунку обрано Angular.

У контексті розроблюваного вебзастосунку клієнтська частина, перш за все, відповідає за відображення інтерфейсу користувача. Це включає в себе меню, сторінки з інформацією, форми для заповнення та інші елементи, необхідні для взаємодії з користувачем. Крім того, клієнтська частина реалізовує логіку взаємодії. Вона оброблює дії користувача, відповідає за валідацію введених даних та повідомлення користувача про будь-які помилки.

Angular дозволяє створювати компоненти графічного інтерфейсу, які можна перевикористовувати та легко підтримувати. Компоненти утворюють ієрархію, де кожен елемент відповідає за певну частину функціональності або відображення сторінки. Механізмом для переходу між графічними компонентами та керуванням стану додатку є модуль маршрутизації.

Ще однією з складових клієнтської частини є сервіси для взаємодії з API. Вони, використовуючи HTTP-клієнт, надають системі інтерфейс для взаємодії з API серверної частини застосунку, дозволяючи виконувати різні операції, такі як отримання, оновлення, створення та видалення ресурсів на сервері.

Дані, отримані з серверної частини, або тимчасові дані, необхідні для роботи на клієнтському пристрої, зберігаються у локальному сховищі. Це

дозволяє зберігати налаштування користувача або тимчасово збережені дані форм без необхідності постійного звернення до сервера.

Узагальнюючи, ці складові створюють потужну та функціональну клієнтську частину вебзастосунку, яка забезпечує зручний та ефективний інтерфейс для користувачів та взаємодію з серверною частиною.

Серверна частина, що базується на Spring Framework, відповідає за обробку запитів, отриманих від клієнтської частини, виконання бізнес-логіки, збереження та отримання даних з баз даних, аутентифікацію та авторизацію користувачів. Розроблювана серверна частина містить такі складові:

- REST контролери. Центральні елементи серверної частини, що приймають HTTP-запити від клієнта та оброблюють їх. Саме вони розділяють серверну частину застосунку на логічні ресурси, реалізовують логіку, необхідну для виконання запитів, та повертають результати у вигляді відповідей.
- Ланцюжки фільтрів Spring Security. Елементи захисту від несанкціонованого доступу та атак. Встановлюють правила доступу до ресурсів і виконують різноманітні перевірки та фільтрації запитів до сервера.
- Сервіс авторизації. Сервіс, що забезпечує перевірку і підтвердження ідентифікаційних даних користувача. Він виконує авторизацію, визначає рівень доступу та надає користувачам необхідні права для доступу до ресурсів.
- Доменні сервіси. Сервіси, що містять бізнес-логіку додатку та виконують операції над даними. Відповідають за оброблення та аналіз даних, що надходять від клієнтів, та реалізацію бізнес-правил системи.
- Репозиторії для взаємодії з БД. Абстрактні елементи, реалізовані ORM-системою Hibernate, відповідальні за доступ до баз даних та

виконання операцій збереження, оновлення, видалення та пошуку даних.

- Бази даних. Реляційні та нереляційні системи для збереження та організації даних, що використовуються системою. Забезпечують надійність, масштабованість та ефективність роботи з інформацією.
- Модуль міграції баз даних. Інструмент для автоматичного оновлення баз даних при випуску нової версії.
- Брокер повідомлень. Посередник у передачі повідомлень між різними компонентами системи. Забезпечує асинхронну комунікацію та використовується для реалізації черги повідомлень.
- Модуль координації розподілених систем. Важливий і необхідний компонент для коректного функціонування брокера повідомлень. Забезпечує маршрутизацію та управління запитами.
- Сервіс сповіщень. Сервіс, що використовується для відправлення повідомлень користувачам про різноманітні події. Він може включати в себе різні канали сповіщень, такі як електронна пошта, повідомлення в месенджерах та інше.
- Сервіс для взаємодії з API брокера Email та Email клієнт. Набір інструментів для взаємодії з API зовнішнього постачальника електронної пошти. Забезпечує можливість відправлення електронних листів користувачам через зовнішній провайдер електронної пошти.
- Модуль моніторингу та журналювання. Цей модуль відповідає за збір та аналіз журналів подій, а також за моніторинг роботи серверної частини. Він дозволяє виявляти потенційні проблеми в роботі програми та вчасно реагувати на них, а також забезпечує зручний інтерфейс для аналізу подій для адміністраторів системи.

На рис. 3.3 представлена структурна схема вебзастосунку.

Рис. 3.3. Структура схема вебзастосунку

3.3. Архітектура серверної частини застосунку

Серверна частина, розроблюваного програмного застосунку, складається з декількох мікросервісів, кожен з яких виконує окремі CRUD операції та взаємодіє з іншими.

Одним з ключових архітектурних рішень, що сприяє досягненню масштабованості, гнучкості, надійності та ефективній роботі мікросервісів, є розміщення їх в контейнерах.

Контейнери забезпечують ізоляцію середовищ виконання, що дозволяє кожному мікросервісу працювати незалежно від інших, що забезпечує слабку зв'язність між ними. Іншою важливою перевагою контейнеризації є підвищена масштабованість. Контейнери дозволяють легко розгортати нові інстанції мікросервісів, коли це необхідно для задоволення попиту. В подальшому можливе використання оркестраційних платформи, таких як Kubernetes, які автоматизують розподіл контейнерів та забезпечують моніторинг і управління їх станом.

Серед основних мікросервісів можна виділити: log, car, user та notification. Кожен с цих мікросервісів має налаштовані порти для доступу до відповідного REST сервісу, який відповідає за операції з певними сутностями (записами технічного обслуговування, автомобілями, СТО, користувачами та сповіщеннями відповідно).

Загалом можна виділити чотири типи контейнерів:

- Перший тип контейнерів містить ресурсні мікросервіси для CRUD операцій. Кожен з цих контейнерів включає RESTful додаток з певним налаштуванням портів для доступу ззовні.
- Другий тип контейнерів призначений для баз даних. Контейнер з базою даних MySQL забезпечує зберігання структурованих даних, тоді як контейнер з MongoDB – неструктурованих.
- Третій тип контейнерів містить брокер повідомлень і сервіс оркестрації. Kafka використовується для асинхронного надсилання сповіщень і збереження повідомлень, тоді як Zookeeper забезпечує координацію розподілених компонентів Kafka.
- Четвертий тип контейнерів включає сервіси для збору та перегляду системних логів. Logstash обробляє та зберігає логи, Elasticsearch забезпечує пошук і аналіз логів, а Kibana надає інструменти для

візуалізації та аналізу логів, що зберігаються в Elasticsearch.

Важливо зазначити, що всі компоненти мікросервісної архітектури розроблюваного застосунку взаємодіють через мережу SII-NET з мережевим драйвером типу Bridge.

Драйвер цього типу, по суті, є мостом між контейнером та хост-машиною. Зазвичай, такий тип драйверів використовується для забезпечення виконання сервісів в автономних контейнерах, між якими передбачена взаємодія.

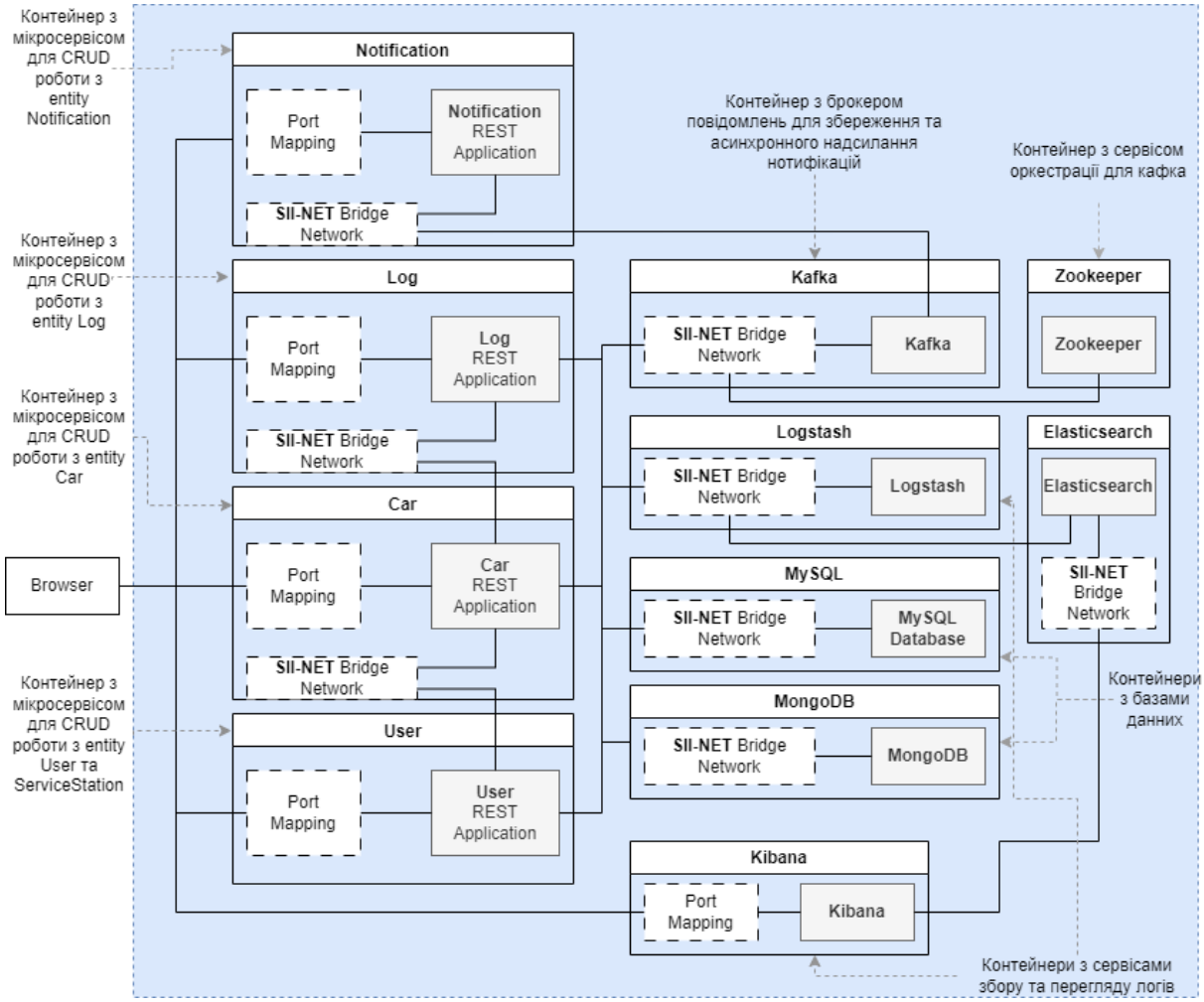


Рис. 3.4. Структурна схема контейнерів серверної частини вебзастосунку

3.4. Структура бази даних

Застосунок складається з шести основних сутностей: Users (Користувачі), Cars (Автомобілі), Notifications (Сповіщення),

Logs (Записи технічного обслуговування), Docs (Документи), Service Stations (СТО). Відповідно кожній з цих сутностей описано таблиці для збереження в базі даних. Структурну схему бази даних наведено на рис. 3.5.

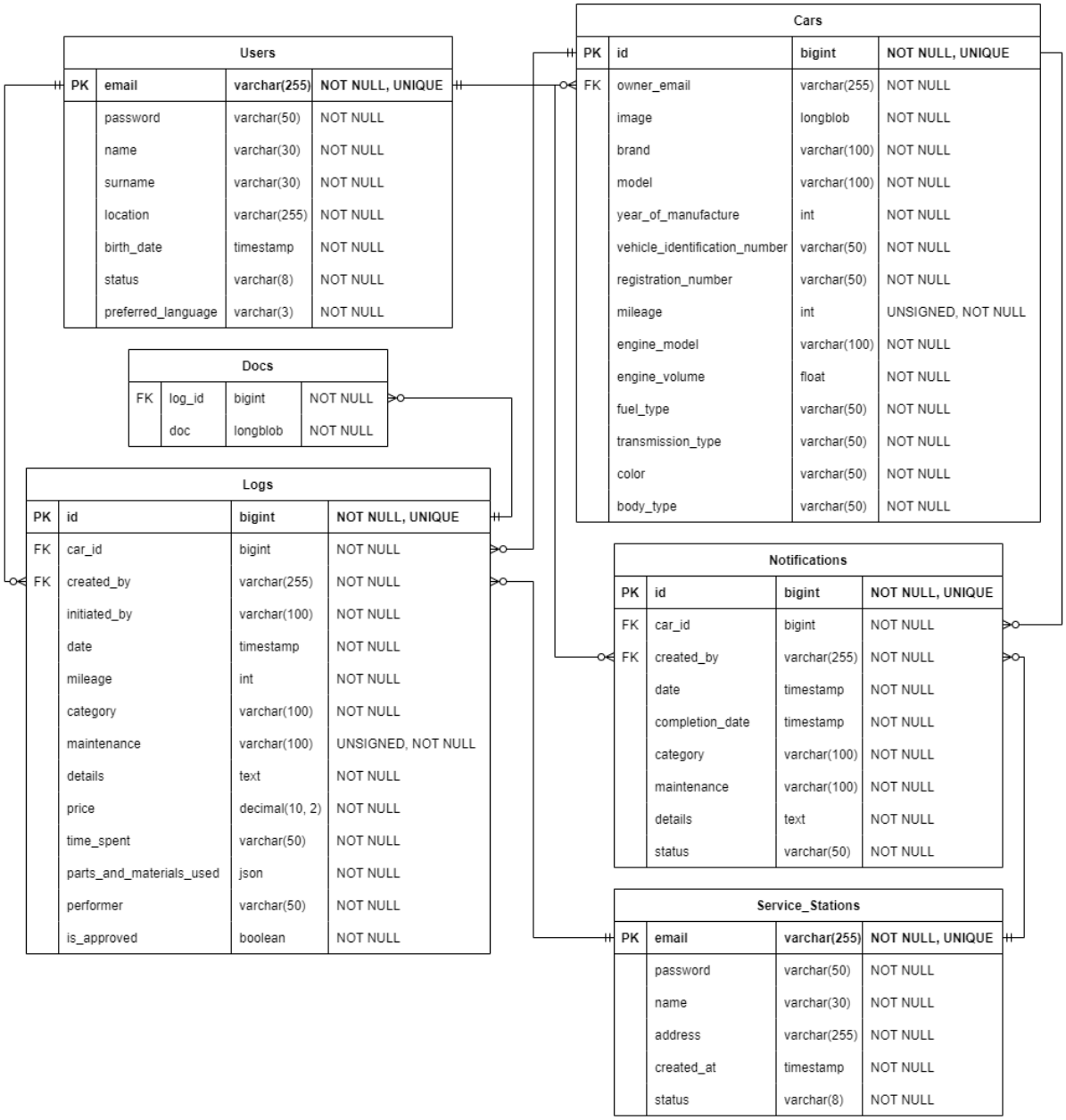


Рис. 3.5. Структурна схема бази даних

Кожна таблиця має свої поля та зв’язки між іншими таблицями для зберігання та управління даними. Далі описаний перелік таблиць та всі їх поля.

1) Users (Користувачі):

- Email – електронна пошта та унікальний ідентифікатор користувача, що використовується для аутентифікації та зв'язку з іншими таблицями.
- Password – пароль для входу користувача, який зберігається в захешованому форматі.
- Name – ім'я користувача.
- Surname – прізвище користувача.
- Location – місце розташування користувача (наприклад, місто або адреса).
- Birth_date – дата народження користувача, може використовуватись для вікових перевірок або персоналізації.
- Status – статус користувача (активний або заблокований).
- Preferred_language – обрана користувачем мова інтерфейсу.

2) Cars (Автомобілі):

- Id – унікальний ідентифікатор автомобіля.
- Owner_email – електронна пошта власника автомобіля та зовнішній ключ для зв'язку з таблицею Users.
- Image – зображення автомобіля.
- Brand – марка автомобіля.
- Model – модель автомобіля.
- Year_of_manufacture – рік виробництва автомобіля.
- Vehicle_identification_number – VIN (номер кузова) автомобіля.
- Registration_number – реєстраційний номер автомобіля.
- Mileage – пробіг автомобіля.
- Engine_model – модель двигуна.
- Engine_volume – об'єм двигуна.
- Fuel_type – тип палива.
- Transmission_type – тип трансмісії.

- Color – колір автомобіля.
- Body_type – тип кузова.

3) Notifications (Сповіщення):

- Id – унікальний ідентифікатор сповіщення.
- Car_id – ідентифікатор автомобіля та зовнішній ключ для зв'язку з таблицею Cars.
- Created_by – електронна пошта користувача, який створив сповіщення та зовнішній ключ для зв'язку з таблицею Users.
- Date – дата створення сповіщення.
- Completion_date – дата завершення дії або обслуговування, зазначеного в сповіщенні.
- Category – категорія сповіщення.
- Maintenance – тип обслуговування.
- Details – деталі сповіщення, більш розгорнута інформація.
- Status – статус сповіщення (відкрито або завершено).

4) Logs (Записи технічного обслуговування):

- Id – унікальний ідентифікатор журналу.
- Car_id – ідентифікатор автомобіля та зовнішній ключ для зв'язку з таблицею Cars.
- Created_by – електронна пошта користувача, який створив запис та зовнішній ключ для зв'язку з таблицею Users.
- Initiated_by – ім'я або ідентифікатор особи або СТО, яка ініціювала дію.
- Date – дата створення запису.
- Mileage – пробіг автомобіля на момент створення запису.
- Category – категорія дії.
- Maintenance – тип обслуговування.
- Details – деталі дії або обслуговування.
- Price – вартість послуги або ремонту.

- Time_spent – витрачений час на обслуговування або ремонт.
- Parts_and_materials_used – список використаних запчастин та матеріалів у форматі JSON.
- Performer – виконавець робіт.
- Is_approved – поле, що показує, чи була підтверджена дія чи обслуговування (true/false).

5) Docs (Документи):

- Log_id – ідентифікатор журналу та зовнішній ключ для зв’язку з таблицею Logs.
- Doc – документ, пов'язаний з журналом, зберігається у форматі longblob.

6) Service_Stations (СТО):

- Email – унікальний ідентифікатор станції.
- Password – пароль для входу.
- Name – назва станції.
- Address – адреса станції.
- Created_at – дата створення запису про станцію.
- Status – статус станції (активна, неактивна).

3.5. Опис реалізованих алгоритмів функціонування системи

Важливою функціональністю для користувачів є можливість перегляду статистики витрат.

З цією метою під час виконання даного дипломного проєкту було створено алгоритм і його реалізацію у вигляді функції для збору та обробки даних про витрати за певний період. Візуалізація цих даних відбувається у вигляді лінійного графіка за допомогою динамічної прив’язки до графічного компонента, а також додавання трендової лінії для відображення загальної тенденції витрат. На рис. 3.6. наведено алгоритм обчислення графіку витрат з лінією тренду у вигляді блок-схеми.

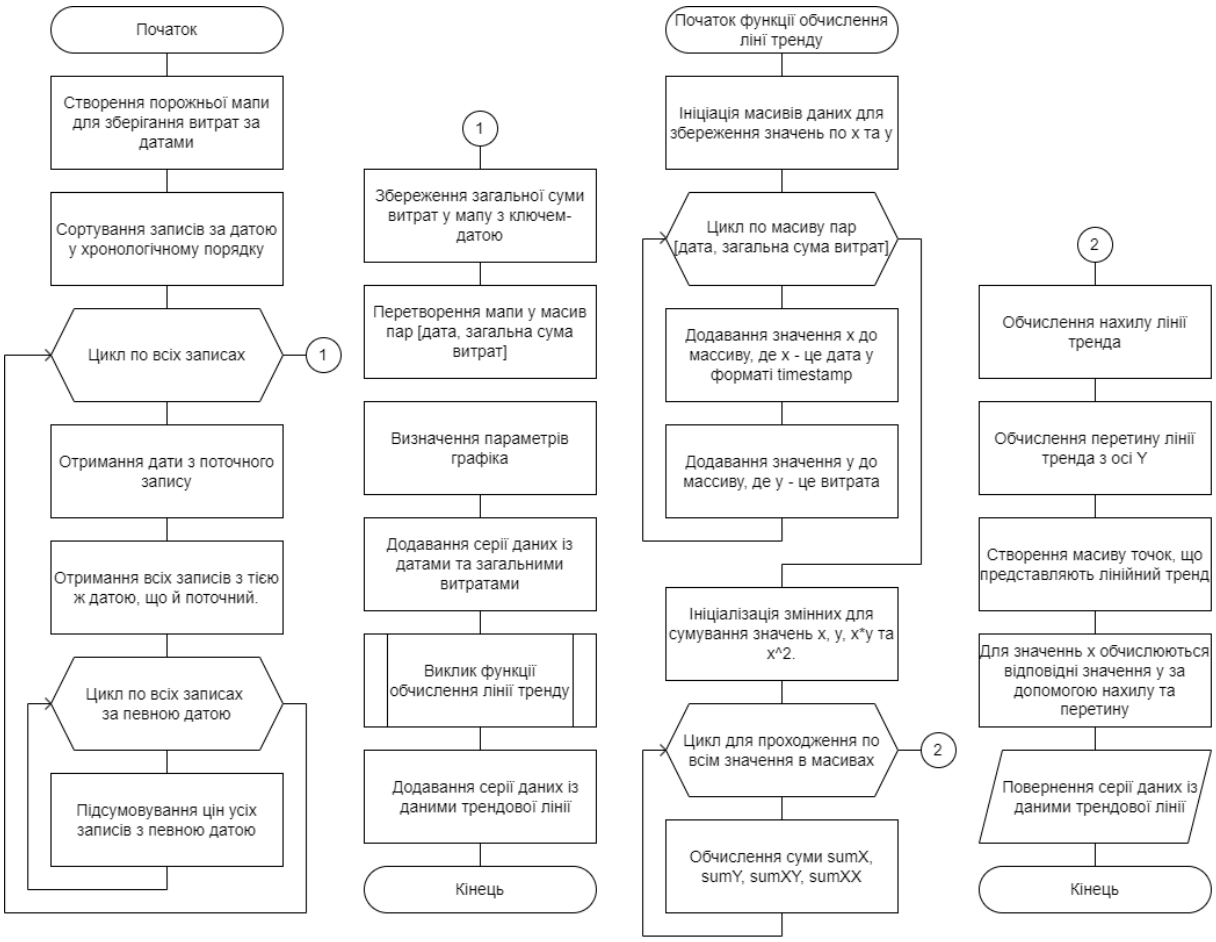


Рис. 3.6. Блок-схема алгоритму обчислення графіку витрат з лінією тренду

Алгоритм починається з ініціалізації порожньої мапи для зберігання суми витрат відповідно до певної дати. Це дозволяє зручно агрегувати дані та проводити подальшу обробку. Далі всі записи витрат сортуються за датою у хронологічному порядку, що забезпечує правильну послідовність даних для подальшого аналізу та візуалізації. Кожен запис обробляється для визначення дати, і для кожної дати підсумовуються витрати, враховуючи всі записи з тією ж датою.

Після обчислення загальної суми витрат та збереження її у мапі, де ключем є дата, а значенням – сума витрат, мапа перетворюється на масив пар, де кожна пара складається з дати та відповідної суми витрат. Це дозволяє легко інтегрувати дані у графік.

На основі збережених у масив пар даних до графіку додаються серії даних, що включають дати та загальні витрати. В кінці додається лінія

тренду, яка розраховується за допомогою окремого алгоритму та показує загальну тенденцію змін витрат.

Алгоритм, призначений для обчислення лінії тренду, яка найкраще описує зміни витрат у часі на основі методу найменших квадратів, починається з перетворення вхідних даних. Дати з вхідного масиву перетворюються у формат timestamp для зручності обчислень, а витрати зберігаються у масиві yValues. Далі визначається кількість елементів у масивах xValues та yValues і ініціалізуються змінні для сумування значень $\sum x\ (sumX),\ \sum y\ (sumY),\ \sum x\ \cdot y\ (sumXY),\ \sum x^2\ (sumXX)$.

Циклічне проходження всіх значень в масивах xValues та yValues дозволяє обчислити ці суми. Відповідно до алгоритму, формули для обчислення сум є наступними:

$$\sum x = \sum_{i=1}^{n} x_i, \tag{3.1}$$

$$\sum y = \sum_{i=1}^{n} y_i, \tag{3.2}$$

$$\sum x \cdot y = \sum_{i=1}^{n} x_i \cdot y_i, \tag{3.3}$$

$$\sum x^2 = \sum_{i=1}^{n} x_i^2. \tag{3.4}$$

Нахил лінії тренду (slope) обчислюється на основі отриманих сум за допомогою формули:

$$slope = \frac{n \cdot \sum(x \cdot y) - \sum x \cdot \sum y}{n \cdot \sum x^2 - (\sum x)^2}. \tag{3.5}$$

Перетин (intercept) лінії тренду з віссю Y обчислюється за формулою:

$$slope = \frac{\sum y - slope \cdot \sum x}{n}. \tag{3.6}$$

Наступним етапом є створення масиву значень, що представляють лінійну тенденцію. Для кожного значення x з xValues обчислюється відповідне значення y за допомогою нахилу та перетину:

$$y = slope\ \cdot x + intercept. \tag{3.7}$$

Пара [x, y] додається до масиву результатів, при цьому значення округлюються до двох десяткових знаків. Після обробки всіх значень функція повертає масив точок, що представляють лінійну тенденцію.

3.6. Висновки до розділу 3

У ході роботи над третім розділом пояснювальної записки розглянуто структурно-алгоритмічну організацію застосунку.

Спочатку був проведений аналіз вимог до розроблюваного вебзастосунку, де були визначені функціональні та нефункціональні вимоги до системи, а також описані основні сценарії використання застосунку різники категоріями користувачів.

Далі було виконано аналіз структури системи, визначено основні компоненти та модулі, такі як серверна частина, клієнтська частина, бази даних, брокер повідомлень і сервіс сповіщень. Деталізовано роль кожного з компонентів у загальній архітектурі системи. Розглянуто механізм міграції баз даних для автоматичного оновлення при випуску нових версій системи та архітектурне рішення для досягнення масштабованості та надійності системи.

Для забезпечення зберігання та обробки даних було розроблено модель даних і визначено формати зберігання даних в базі даних. Побудовано ER-діаграму.

Розроблено та проаналізовано основні алгоритми обробки даних, включаючи алгоритми для обчислення лінійних тенденцій та збору статистики витрат. Представлено блок-схеми, що демонструють процеси обробки даних та взаємодії між компонентами системи.

4. ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Реалізація серверної частини

При розробленні архітектури та виборі технологій було приділено особливу увагу забезпеченню високої продуктивності, надійності та масштабованості для застосунку.

Серверна частина побудована на основі сучасного веб-фреймворку Spring, який надає потужну інфраструктурну підтримку для розроблення застосунків. Використання цього фреймворку дозволило розділити серверну частину на декілька логічних шарів. Це спрощує управління залежностями, забезпечує безпеку та покращує можливість до тестування коду.

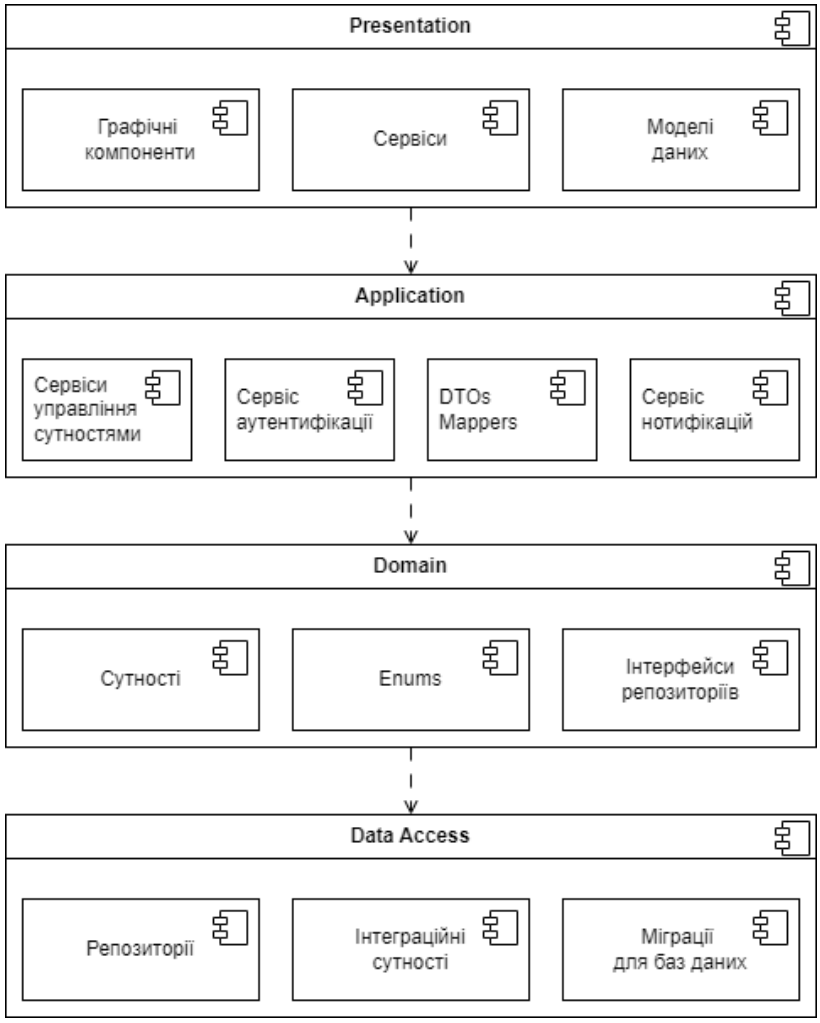


Рис. 4.1. Діаграма декомпозиції застосунку на логічні шари

Діаграма, продемонстрована на рис. 4.1, ілюструє розподіл застосунку на логічні шари, де рівень застосунку (Application), рівень домену (Domain) та рівень доступу до даних (Data Access) відносяться до серверної частини застосунку. Розглянемо кожен с цих логічних рівнів:

- Рівень презентації. Рівень презентації відповідає за логіки інтерфейсу користувача та його взаємодії з клієнтським застосунком на Angular. Він служить мостом між браузером користувача та програмою на стороні сервера.
- Програмний рівень. Рівень, що містить бізнес-логіку програми та організовує потік даних між рівнем презентації та домену, забезпечуючи застосування бізнес-правил.
- Рівень домену. Рівень інкапсуляції стану та поведінки сутностей. Відповідає за представлення сутностей для рівня доступу даних. Забезпечує валідність даних.
- Рівень доступу даних. Відповідає за збереження та отримання даних із бази даних. Забезпечує абстракцію від сховища даних, приховуючи деталі операцій з керування даними.

Основними компонентами серверної частини є REST-сервіси, які оброблюють HTTP-запити від клієнта та повертають відповіді. Для реалізації REST-сервісів використовується Spring Boot. Особливістю Spring Boot є те, що він, крім забезпечення швидкого старту проєкту та автоматизації конфігурації, надає для управління та розгортання контейнера сервлетів Tomcat. Tomcat забезпечує середовище виконання Java-коду на стороні сервера та відповідає за управління життєвим циклом сервлетів. Коннектор контейнера сервлетів приймає HTTP-запити від клієнтів, обробляє їх та передає до відповідного сервлету виконання. В даному випадку точкою, з якої починається відпрацювання сервлету, є REST-контролери.

Для підтримки створення RESTful вебсервісів було використано модуль Spring Web. Цей модуль надає зручні інструменти для оброблення

HTTP-запитів і створення відповідей, включаючи анотації, необхідні для створення REST-контролерів, такі як @RestController і @RequestMapping.

Для забезпечення інтеграції з Java Persistence API (JPA) та виконання CRUD операції було використано модуль Spring Data JPA. Основні компоненти цього модуля включають інтерфейси Repository для роботи з базою даних та Entity класи для моделювання таблиць бази даних.

Важливим етапом при розробленні серверної частини є забезпечення захисту від несанкціонованого доступу та атак. З цією метою було використано модуль Spring Security. Модуль Spring Security надає потужні засоби для аутентифікації та авторизації користувачів. Компоненти цього модуля дозволяють швидко та зручно налаштовувати ланцюжки фільтрів для перевірки запитів та встановлення правил доступу. А інформація, збережена про сесію користувача, зберігається в SecurityContext.

Централізованим сервісом для проходження етапу авторизації та аутентифікації в такому випадку обрано сервіс для обробки користувацьких даних. При авторизації цей сервіс перевіряє ідентифікаційні дані користувача та визначає його рівень доступу до ресурсів. Як результат, до клієнтської частини в якості відповіді на запит авторизації повертається токен доступу. Важливо зазначити, що цей токен доступу має період існування, і після його проходження він стає недійсним.

Розділення серверної частини на логічні рівні реалізовано за допомогою одного з принципів забезпечення слабкої зв’язності коду, а саме інверсії контролю (IoC) [27]. Реалізація інверсії контролю для фреймворку Spring називається ін’єкція залежностей (Dependency Injection).

Одиницею для ін'єкції є бін (bean). Бін – це завершений програмний елемент с певною бізнес-функцією або внутрішньою функцією Spring. Відповідальність за життєвий цикл біна повністю лежить на контейнері бінів – ApplicationContext.

4.2. Реалізація клієнтської частини

Клієнтська частина вебзастосунку є ключовим елементом, оскільки саме вона забезпечує взаємодію користувача з системою.

На етапі планування системи було обрано фреймворк Angular. Використання компонентного підходу до розроблення клієнтської частини дозволило розподілити відповідальність за певні функціональні можливості між різними компонентами.

Для забезпечення адаптивності інтерфейсу і, відповідно, коректного його відображення та зручності використання на різних пристроях, таких як персональні комп’ютери, планшети та телефони було використано медіазапити.

4.2.1. Маршрутизація

Для забезпечення навігації та маршрутизації між компонентами використаний модуль AppRoutingModule. В цьому модулі описано список маршрутів з прив’язкою до URL-адреси та компоненту, який буде опрацьовувати відповідний маршрут.

Таблиця 4.1

Маршрутизація у клієнтській частині

Вебсторінка	Шлях	Назва компоненту
Сторінка авторизації	/sign-in	SignInComponent
Сторінка реєстрації	/sign-up	SignUpComponent
Профіль користувача	/profile	ProfileComponent
Сторінка перегляду автомобілів	/cars	CarsComponent
Календар	/calendar	CalendarComponent
Сторінка СТО	/service	ServiceComponent
Панель моніторингу	/dashboard	DashboardComponent

4.3. Тестування програмного забезпечення

Одним з ключових етапів розроблення програмного забезпечення є тестування. Відповідно до функціональних вимог було створено перелік тестових сценаріїв, описаних в табл. 4.2, і з метою виявлення потенційних дефектів основних функцій системи проведено димове та White Box тестування.

Таблиця 4.2

Сценарії тестування вебзастосунку

№	Код вимоги	Дії	Очікуваний результат
1	F-1	1. Відкрити сторінку реєстрації. 2. Коректно заповнити всі поля. 3. Натиснути "Зареєструватися".	Реєстрація успішна, користувача переадресовано на головну сторінку.
2	F-2	1. Відкрити сторінку реєстрації. 2. Ввести некоректний формат електронної адреси. 3. Натиснути "Зареєструватися".	З'являється попередження про некоректний формат електронної адреси, процес реєстрації заблоковано.
3	F-3	1. Відкрити сторінку реєстрації. 2. Не заповнити одне з обов'язкових полів. 3. Натиснути "Зареєструватися".	З'являється попередження про незаповнене поле, процес реєстрації заблоковано.
4	F-4	1. Відкрити сторінку авторизації. 2. Ввести правильну електронну адресу та пароль. 3. Натиснути "Увійти".	Авторизація успішна, користувача переадресовано на головну сторінку.
5	F-5	1. Відкрити сторінку авторизації. 2. Ввести неправильну електронну адресу або пароль. 3. Натиснути "Увійти".	З'являється попередження про неправильні дані, процес авторизації заблоковано.

Продовження табл. 4.2

6	F-6	1. Авторизуватись. 2. Перейти на сторінку профілю. 3. Натиснути кнопку редагування профілю. 4. Змінити ім'я та натиснути "Зберегти".	Ім'я успішно змінено, відображено оновлену інформацію.
7	F-7	1. Авторизуватись. 2. Перейти на сторінку профілю. 3. Натиснути кнопку редагування профілю. 4. Не заповнити одне з полів. 5. Натиснути "Зберегти".	З'являється попередження про незаповнене поле, процес збереження заблоковано.
8	F-8	1. Авторизуватись. 2. Перейти на сторінку профілю. 3. Натиснути кнопку зміни паролю. 4. Ввести новий пароль та його підтвердження. 5. Натиснути "Зберегти".	Пароль успішно змінено.
9	F-9	1. Авторизуватись. 2. Перейти на сторінку профілю. 3. Натиснути кнопку зміни паролю. 4. Ввести новий пароль, що не відповідає безпековим вимогам. 5. Натиснути "Зберегти".	З'являється попередження про ненадійний пароль, процес зміни паролю заблоковано.
10	F-10	1. Авторизуватись. 2. Натиснути "Вийти".	Користувач вийшов із системи. Користувача переадресовано на сторінку авторизації.
11	F-11	1. Авторизуватись. 2. Змінити мову інтерфейсу на англійську. 3. Вийти та зайти знову.	Інтерфейс системи відображається англійською мовою.

Продовження табл. 4.2

12	F-12	1. Авторизуватись. 2. Перейти на сторінку транспортних засобів.	Відображається список всіх транспортних засобів користувача.
13	F-13	1. Авторизуватись. 2. Перейти на сторінку транспортних засобів. 3. Натиснути кнопку додавання транспортного засобу. 4. Заповнити всі поля правильно. 5. Натиснути "Додати".	Транспортний засіб успішно додано, відображається оновлений список транспортних засобів.
14	F-14	1. Авторизуватись. 2. Перейти на сторінку транспортних засобів. 3. Натиснути кнопку додавання транспортного засобу. 4. Ввести від'ємний пробіг. 5. Натиснути "Додати".	З'являється попередження про некоректні дані, процес додавання транспортного засобу заблоковано.
15	F-15	1. Авторизуватись. 2. Перейти на сторінку профілю. 3. Натиснути кнопку передачі історії обслуговування. 4. Ввести коректну електронну адресу нового власника. 5. Натиснути "Передати".	Історія обслуговування успішно передана новому власнику.
16	F-16	1. Авторизуватись. 2. Перейти на сторінку профілю. 3. Натиснути кнопку передачі історії обслуговування. 4. Ввести некоректну електронну адресу нового власника. 5. Натиснути "Передати".	З'являється попередження про некоректну електронну адресу, процес передачі історії обслуговування заблоковано.

Продовження табл. 4.2

17	F-17	1. Авторизуватись. 2. Перейти на сторінку профілю. 3. Вибрати транспортний засіб та натиснути "Видалити". 4. Підтвердити видалення.	Транспортний засіб успішно видалено зі списку.
18	F-18	1. Авторизуватись. 2. Перейти на сторінку транспортних засобів. 3. Перейти на сторінку записів технічного обслуговування транспортного засобу.	Відображається список операцій, згрупованих за категоріями. Верифіковані записи мають особливу відмітку та назву СТО.
19	F-19	1. Авторизуватись. 2. Перейти на сторінку транспортних засобів. 3. Перейти на сторінку записів технічного обслуговування транспортного засобу. 4. Натиснути кнопку додавання запису обслуговування. 5. Заповнити всі поля правильно. 6. Натиснути "Додати".	Запис обслуговування успішно додано, відображається в списку операцій.
20	F-20	1. Авторизуватись. 2. Перейти на сторінку транспортних засобів. 3. Перейти на сторінку записів технічного обслуговування транспортного засобу. 4. Натиснути кнопку додавання запису обслуговування. 5. Ввести некоректні дані. 6. Натиснути "Додати"..	З'являється попередження про некоректні дані, процес додавання запису заблоковано.

Продовження табл. 4.2

21	F-21	1. Авторизуватись. 2. Перейти на сторінку транспортних засобів. 3. Перейти на сторінку записів технічного обслуговування транспортного засобу. 4. Вибрати запис та натиснути "Видалити". 5. Підтвердити видалення.	Запис обслуговування успішно видалено зі списку.
22	F-22	1. Авторизуватись. 2. Перейти на сторінку календаря.	Відображається календар подій. Відображається список запланованих події з можливістю додавання нових подій.
23	F-23	1. Авторизуватись. 2. Перейти на сторінку календаря. 3. Натиснути кнопку додавання нової події. 4. Зберегти подію.	Подію успішно додано, відображено у календарі.
24	F-24	1. Авторизуватись як працівник СТО. 2. Ввести коректну електронну пошту користувача.	Відображається перелік автомобілів користувача.
25	F-25	1. Авторизуватись як працівник СТО. 2. Ввести некоректну електронну пошту користувача.	Відображається попередження про некоректні дані.

4.4. Напрямки щодо подальшого вдосконалення

Тематика даного дипломного проєкту не обмежена створенням лише описаних функціональних можливостей.

Для подальшого вдосконалення системи ведення обліку технічного обслуговування транспортних засобів варто звернути увагу на такі модифікації:

- додавання можливості автоматичного генерування звітів у форматах PDF або XLSX;
- можливість для СТО переглядати статистику;
- можливість отримання сповіщень в популярних месенджерах, таких як Telegram, Viber, WhatsApp або SMS;
- інтеграції з сучасними сервісами OAuth 2.0 авторизації, такими як Facebook, Google, Twitter;
- інтеграція з іншими календарями, такими як Google Calendar та Microsoft Outlook;
- додавання підтримки більшої кількості мов інтерфейсу;
- додавання мапи найближчих СТО.

Представлені модифікації суттєво покращать цінність даного проєкту в очах користувачів.

4.5. Висновки до розділу 4

У даному розділі дипломного проєкту розглянуто особливості реалізації програмного забезпечення для ведення обліку технічного обслуговування транспортного засобу, що включає огляд реалізації серверної частини та функціональних можливостей клієнтської частини системи. Було оглянуто користувацький інтерфейс та описано основні сторінки та діалогові вікна. Описано основні сценарії використання та проведено димове тестування вебзастосунку. Результати, отримані під час тестування, підтверджують готовність системи до використання. Також для розробленого застосунку було визначено шляхи для подальшого вдосконалення.

ВИСНОВКИ

Під час написання даного дипломного проєкту проведено детальний аналіз існуючих рішень, що присутні на ринку, та розглянуто їх функціональні можливості. Проведений аналіз показав, що наявні рішення мають певні недоліки, які не дозволяють їм повністю задовольнити потреби користувачів. І саме тому існує певний простір для створення нового застосунку, який би поєднував переваги існуючих рішень та усував їхні недоліки.

На основі порівняльного аналізу технологічних стеків було обрано найефективніші інструменти для розробки серверної та клієнтської частин застосунку. Також було обрано бази даних, які забезпечували б надійне та ефективне зберігання даних.

Для досягнення масштабованості та надійності було детально розглянуто основні модулі та компоненти системи, а також механізми для досягнення масштабованості. Для компонентів визначено роль у загальній архітектурі та розроблено модель даних, відображену у побудованій ER-діаграмі. Проаналізовано та описано функціональні та нефункціональні вимоги до програмного забезпечення.

Реалізація програмного забезпечення передбачала опис особливостей, а саме опис реалізації серверної та клієнтської частин застосунку, сторінок та сценаріїв використання, основних алгоритмів.

Проведене тестування підтвердило готовність системи до використання.

Результатом виконання даного дипломного проєкту є інноваційне рішення, яке забезпечує потреби користувачів та поєднує переваги існуючих рішень та усуває їхні недоліки.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Simply Auto: Car maintenance and Mileage Tracker app [Електронний ресурс]. — Режим доступу: https://simplyauto.app — (18.04.2024).
2. CarDiary - Everything for your vehicle [Електронний ресурс]. — Режим доступу: https://www.car-diary.net/en — (18.04.2024).
3. MyCarLog – diary of your car! [Електронний ресурс]. — Режим доступу: https://mycarlogapp.com — (18.04.2024).
4. Simply Auto: Mileage Tracker [Електронний ресурс]. — Режим доступу: https://apps.apple.com/us/app/simply-auto-mileage-tracker/id893278325 — (18.04.2024).
5. Simply Auto: Car Maintenance [Електронний ресурс]. — Режим доступу: https://play.google.com/store/apps/details?id=mrigapps.andriod.fuelcons&hl=uk — (18.04.2024).
6. CarDiary - Дневник на колата [Електронний ресурс]. — Режим доступу: https://play.google.com/store/apps/details?id=car.diary.app — (18.04.2024).
7. CarDiary - Vehicle log book [Електронний ресурс]. — Режим доступу: https://apps.apple.com/us/app/cardiary-vehicle-log-book/id1509773325 — (18.04.2024).
8. MyCarLog: Car management [Електронний ресурс]. — Режим доступу: https://apps.apple.com/ua/app/mycarlog-car-management/id1386377748?l=en — (18.04.2024).
9. Що краще моноліт чи мікросервіси? [Електронний ресурс]. — Режим доступу: https://iampm.club/ua/blog/shho-krashhe-monolit-chi-mikroservisi-yak-obrati-arhitekturu-projektu/ — (18.04.2024).
10. Comparing Apache Kafka, ActiveMQ, and RabbitMQ [Електронний ресурс]. — Режим доступу: https://www.conduktor.io/blog/comparing-apache-kafka-activemq-and-rabbitmq/ — (18.04.2024).
11. Рейтинг мов програмування 2024. TypeScript в трійці лідерів, Python з’являється у всіх нішах, а Rust — улюблена мова [Електронний ресурс].

— Режим доступу: https://dou.ua/lenta/articles/language-rating-2024/ — (18.04.2024).
12. Advantages of Java [Електронний ресурс]. — Режим доступу: https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Java-Advantages-Benefits-Fast-Performance-Simple-Open-Typed-Features-Streams — (18.04.2024).
13. Pros and Cons of Python Programming Language – Serokell [Електронний ресурс]. — Режим доступу: https://serokell.io/blog/python-pros-and-cons — (18.04.2024).
14. Advantages and Disadvantages of Python - Tagline Infotech LLP [Електронний ресурс]. — Режим доступу: https://taglineinfotech.com/advantages-and-disadvantages-of-python — (18.04.2024).
15. The Good and the Bad of C# Programming [Електронний ресурс]. — Режим доступу: https://www.altexsoft.com/blog/c-sharp-pros-and-cons/ — (18.04.2024).
16. Java Spring Pros and Cons [Електронний ресурс]. — Режим доступу: https://www.javatpoint.com/java-spring-pros-and-cons — (18.04.2024).
17. Your relational data. Objectively. - Hibernate ORM [Електронний ресурс]. — Режим доступу: https://hibernate.org/orm/ — (18.04.2024).
18. Liquibase vs Flyway [Електронний ресурс]. — Режим доступу: https://www.baeldung.com/liquibase-vs-flyway — (18.04.2024).
19. ELK Stack: Elasticsearch, Kibana, Beats & Logstash [Електронний ресурс]. — Режим доступу: https://www.elastic.co/elastic-stack — (18.04.2024).
20. Elasticsearch: The Official Distributed Search & Analytics Engine | Elastic [Електронний ресурс]. — Режим доступу: https://www.elastic.co/elasticsearch — (18.04.2024).
21. Logstash: Collect, Parse, Transform Logs [Електронний ресурс]. — Режим доступу: https://www.elastic.co/logstash — (18.04.2024).
22. Kibana: Explore, Visualize, Discover Data [Електронний ресурс]. — Режим

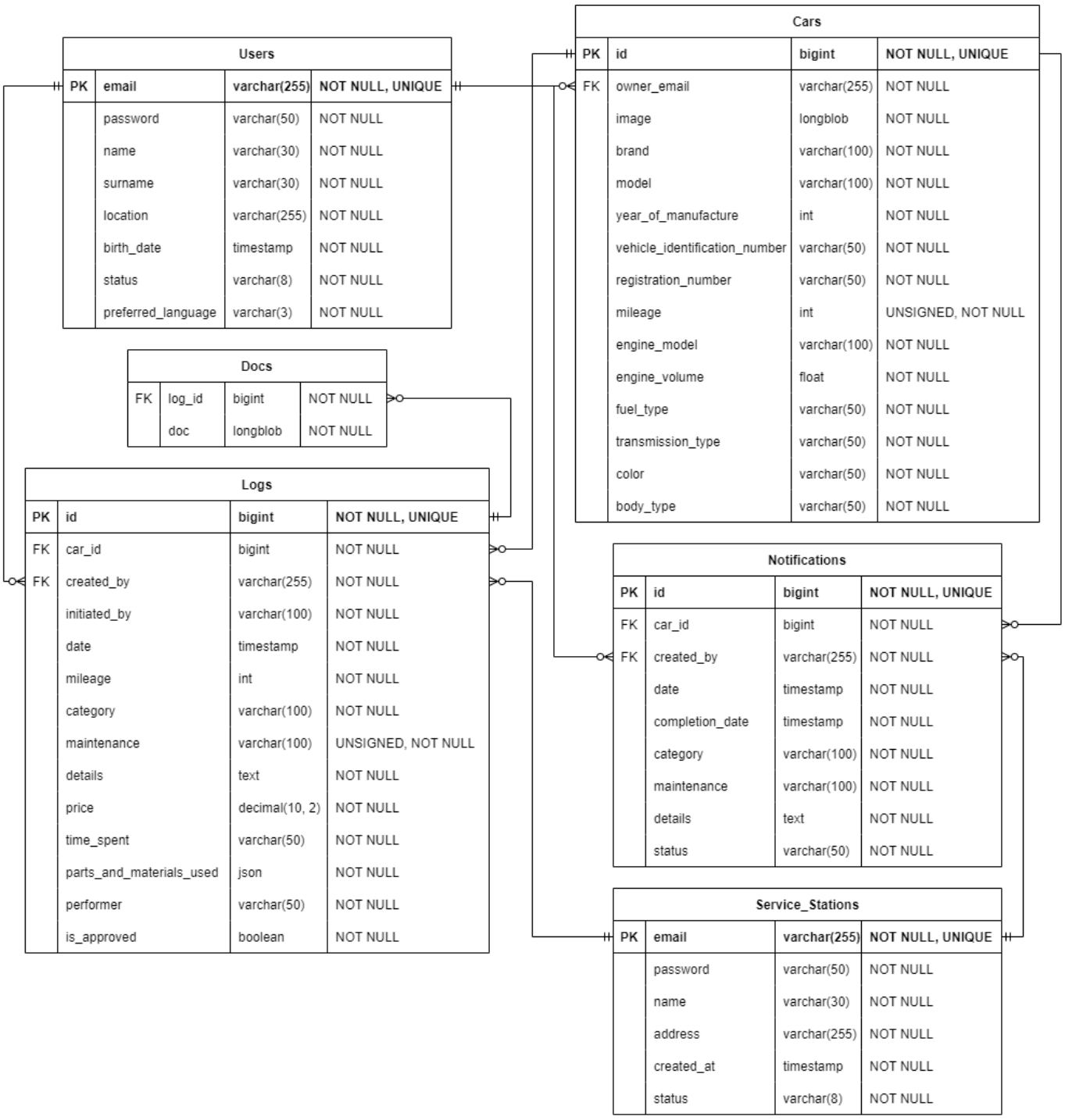
доступу: https://www.elastic.co/kibana — (18.04.2024).

23. Angular vs React vs Vue [Електронний ресурс]. — Режим доступу: https://eluminoustechnologies.com/blog/angular-vs-react-vs-vue/ — (18.04.2024).
24. Angular vs React vs Vue: Which Framework to Choose [Електронний ресурс]. — Режим доступу: https://wpshout.com/angular-vs-vue-vs-react/ — (18.04.2024).
25. Relational vs Nonrelational Databases - Amazon AWS [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/compare/the-difference-between-relational-and-non-relational-databases/ — (18.04.2024).
26. Функціональні та нефункціональні вимоги – Visure Solutions [Електронний ресурс]. — Режим доступу: https://visuresolutions.com/uk/посібник-з-відстеження-управління-вимогами/функціональні-та-нефункціональні-вимоги/ — (18.04.2024).
27. Dependency Injection, Dependency Inversion, IoC Choose [Електронний ресурс]. — Режим доступу: https://medium.com/ssense-tech/dependency-injection-vs-dependency-inversion-vs-inversion-of-control-lets-set-the-record-straight-5dc818dc32d1 — (18.04.2024).

ДОДАТКИ

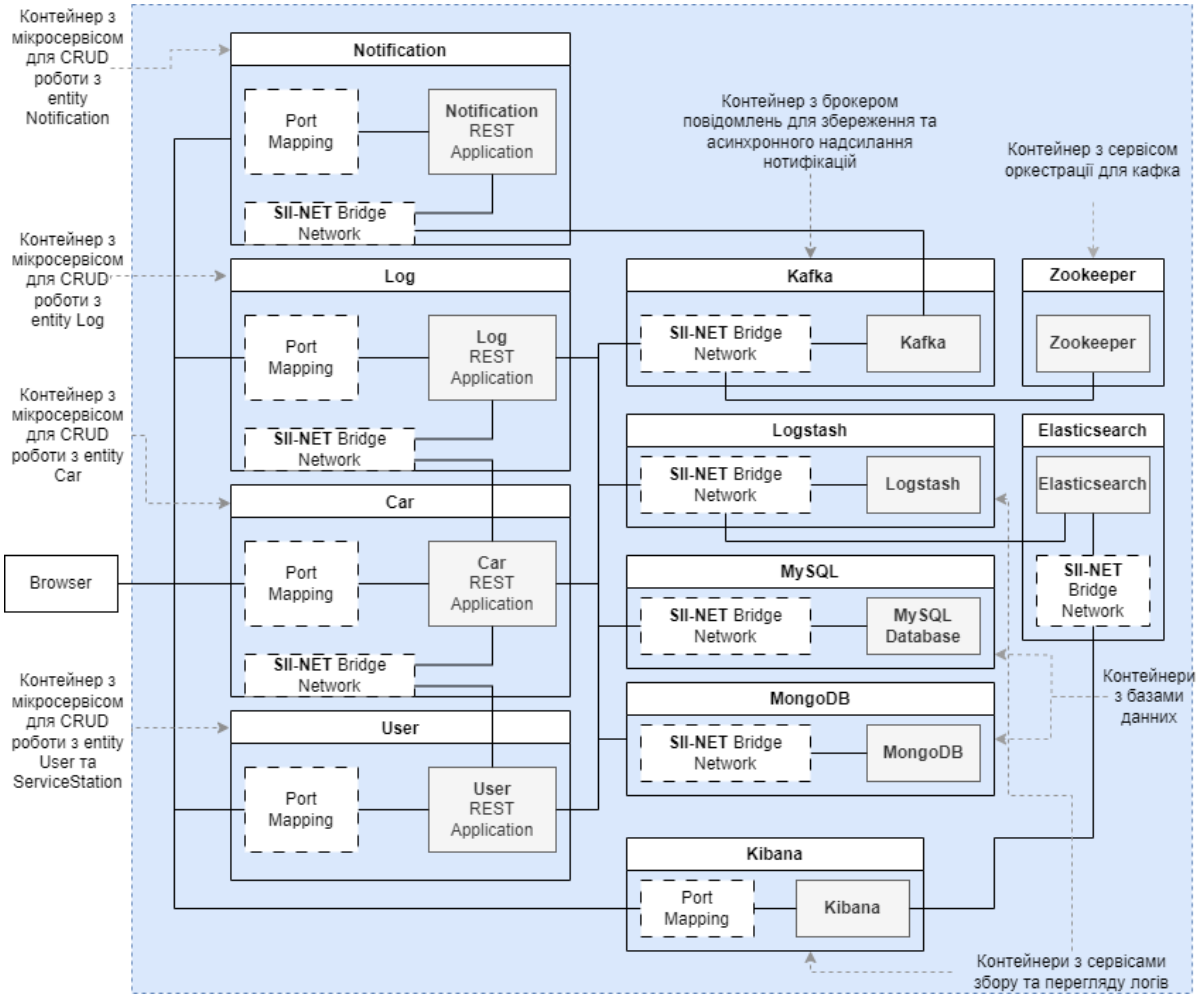
Додаток 1

Копії графічних матеріалів

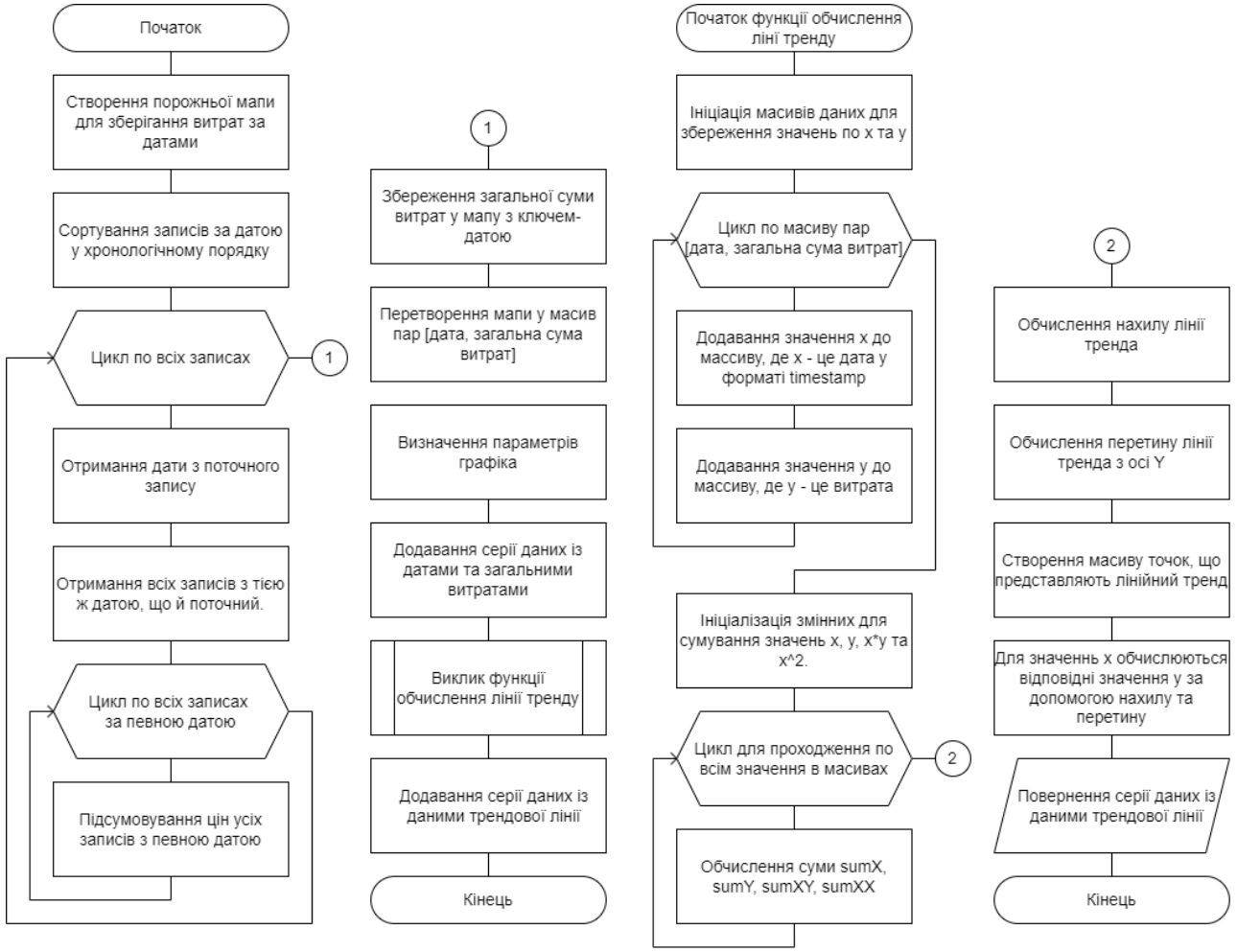


ДП.045440-06-99. Вебзастосунок для ведення обліку технічного стану транспортних засобів. Структурна схема бази даних. ER-діаграма

ДП.045440-07-99. Вебзастосунок для ведення обліку технічного стану транспортних засобів. Структурна схема вебзастосунку. UML-діаграма



Структурна схема контейнерів серверної частини вебзастосунку
Довженко Р.О., група КП-03



Блок-схема алгоритму обчислення графіку витрат з лінією тренду
Довженко Р.О., група КП-03

Додаток 2

Лістинг програми

```
import {Component, ElementRef, OnInit, ViewChild} from '@angular/core';
import {UserService} from "../../services/user.service";
import {CarService} from "../../services/car.service";
import {LogService} from "../../services/log.service";
import * as Highcharts from 'highcharts';
import {Options} from "highcharts/highcharts.src";
import {Car} from "../../entities/car";
import {Log} from "../../entities/log";
import {Router} from "@angular/router";
import {Notification} from "../../entities/notification";
import {NotificationService} from "../../services/notification.service";

interface Category {
  name: string;
  isOpen: boolean;
}

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrl: './dashboard.component.css'
})
export class DashboardComponent implements OnInit {
  @ViewChild('#mat-calendar-header-1-period-label') calendarRef!:
ElementRef;
  private readonly months = ['January', 'February', 'March', 'April',
'May', 'June', 'July', 'August', 'September', 'October', 'November',
'December'];
  public readonly categories: Category[] = [
    {name: "Routine Maintenance (RM)", isOpen: true},
    {name: "Engine Repair", isOpen: true},
    {name: "Brake System", isOpen: true},
    {name: "Suspension and Transmission", isOpen: true},
    {name: "Electrical Equipment and Electronics", isOpen: true},
    {name: "Bodywork", isOpen: true},
    {name: "Cooling System", isOpen: true},
    {name: "Air Conditioning and Heating System", isOpen: true},
    {name: "Tuning and Modifications", isOpen: true},
    {name: "Other Tasks", isOpen: true}
  ]
  private _logs: Log[];
  private _selectedCar: Car | undefined = undefined;
  private _highcharts = Highcharts;
  private _totalCostChart: Options;
  private _totalSpentTime: Options;
  private _categoryChart: Options;
  private _totalCostGraph: Options;
  private _selectedDate: Date = new Date("05/07/2017");
  minDate: Date;
  maxDate: Date;
  weekDays: string[] = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday'];
  _selectedNotifications: Notification[] = [];
  cars: Car[] = [];
    carsForNotification: Map<Notification, Car> = new Map();
  get selectedCar(): Car | undefined {
    return this._selectedCar;
  }
  set selectedCar(value: Car) {
    this._selectedCar = value;
  }
  get highcharts() {
    return this._highcharts;
  }
```

```
  get selectedDate() {
    return this._selectedDate;
  }
  get categoryChart(): Options {
    return this._categoryChart;
  }
  get totalCostGraph(): Options {
    return this._totalCostGraph;
  }
  set selectedDate(value) {
    this._selectedDate = value;
  }
  set highcharts(value) {
    this._highcharts = value;
  }
  get totalCostChart(): Options {
    return this._totalCostChart;
  }
  get totalSpentTime(): Options {
    return this._totalSpentTime;
  }
  constructor(private router: Router, private userService: UserService,
private carService: CarService,
              private logService: LogService, private notificationService:
NotificationService) {
    localStorage.setItem("language", userService.user.preferredLanguage);
    this._logs = logService.logs;
    this._totalCostChart = {} as Options;
    this._totalCostGraph = {} as Options;
    this._totalSpentTime = {} as Options;
    this._categoryChart = {} as Options;
    this._selectedDate = new Date();
    this.minDate = new Date(new Date().getFullYear(), new
Date().getMonth(), 1);
    this.maxDate = new Date(new Date().getFullYear(), new Date().getMonth()
+ 1, 0);
    this.updateNotifications();
    this.loadCars();
  }
  loadCars() {
      this.carService.getCars().subscribe(cars => {
          this.cars = cars;
      });
  }
  ngOnInit(): void {
    this.showTotalCostChart();
    this.showTotalSpentTime();
    this.showCategoryChart();
    this.showTotalCostsGraph();
  }
  updateNotifications() {
    this._selectedNotifications = this.notificationService
      .getByMonth(this._selectedDate.getMonth(),
this._selectedDate.getFullYear())
      .sort((a, b) => {
      return a.date.getTime() - b.date.getTime();
    });
      this._selectedNotifications.forEach(notification => {
          this.carService.getCarById(notification.carId).subscribe(car => {
              this.carsForNotification.set(notification, car);
          })
      });
  }
```

```
  public updateSelectedCar(car: Car) {
    this._selectedCar = car;
    if (car != undefined) {
      this._logs = this.logService.getLogsByCar(car);
      this.ngOnInit();
    } else {
      location.reload();
    }
  }
  public countTotalCosts() {
    return this._logs.reduce((totalCost, log) => totalCost + log.price, 0);
  }
  public countTotalSpentTime() {
    return this._logs.map(log => {
      return this.parseTime(log.timeSpent);
    }).reduce((sum, value) => sum + value, 0);
  }
  public countTotalSpentTimeForCar(car: Car) {
    return this.logService.getLogsByCar(car).map(log => {
      return this.parseTime(log.timeSpent);
    }).reduce((sum, value) => sum + value, 0);
  }
  public convertMinutesToString(minutes: number) {
    const hours = Math.floor(minutes / 60);
    const min = minutes % 60;
    return hours + "hr  " + min + "min";
  }
  private countTotalCostsForCar(car: Car) {
    return this.logService.getLogsByCar(car).reduce((totalCost, log) =>
totalCost + log.price, 0);
  }
  private parseTime(timeString: string) {
    const [hours, minutes] = timeString.split(':').map(Number);
    return hours * 60 + minutes;
  }
  private calculateLinearTrend(data: [string, number][]) {
    const xValues = data.map(value => new Date(value[0]).getTime());
    const yValues = data.map(value => value[1]);
    const n = xValues.length;
    let sumX = 0;
    let sumY = 0;
    let sumXY = 0;
    let sumXX = 0;
    for (let i = 0; i < n; i++) {
      sumX += xValues[i];
      sumY += yValues[i];
      sumXY += xValues[i] * yValues[i];
      sumXX += xValues[i] * xValues[i];
    }
    const slope = (n * sumXY - sumX * sumY) / (n * sumXX - sumX * sumX);
    const intercept = (sumY - slope * sumX) / n;
    return xValues.map(x => Math.round((slope * x + intercept) * 100) /
100);
  }
  showTotalCostsGraph() {
    const map: { [key: string]: number } = {};
    this._logs.sort((a, b) => new Date(a.date).getTime() - new
Date(b.date).getTime())
      .forEach(log => {
        map[new Date(log.date).toDateString()] =
this.logService.getLogsByDate(log.date).reduce((totalCost, log) =>
totalCost + log.price, 0)
      })
```

```
    const data: [string, number][] = Object.entries(map)
      .map(([dateString, totalCost]) => [dateString, totalCost]);
    this._totalCostGraph = {
      chart: {
        type: "line"
      },
      colors: ['#6AF0B7'],
      title: {
        text: ""
      },
      xAxis: {
        categories: data.map(value => value[0])
      },
      legend: {
        enabled: true,
        align: 'right',
        verticalAlign: 'top',
        layout: 'vertical',
        floating: true,
        borderWidth: 1,
        borderRadius: 5,
        backgroundColor: 'rgba(255,255,255,0.7)'
      },
      yAxis: {
        title: {
          text: "Costs"
        }
      },
      plotOptions: {
        line: {
          tooltip: {
            format: '{point.y:.1f}%',
            valueSuffix: '$'
          }
        }
      },
      series: [
        {
          type: "line",
          name: "Date",
          data: data,
          color: 'rgb(44, 175, 254)'
        },
        {
          type: "line",
          name: "Trend",
          data: this.calculateLinearTrend(data),
          color: 'rgb(0, 226, 114)'
        }
      ],
    };
  }
  showTotalCostChart() {
    const data: [string, number][] = [];
    this.cars.forEach(car => {
      data.push([car.brand + " " + car.model,
this.countTotalCostsForCar(car)]);
    });
    this._totalCostChart = {
      chart: {
        type: "pie",
        backgroundColor: '#F3EEFE',
        borderRadius: 12,
        styledMode: false
```

```
      },
      title: {
        text: ""
      },
      xAxis: {
        categories: data.map(value => value[0])
      },
      yAxis: {
        title: {
          text: ""
        }
      },
      legend: {
        enabled: false
      },
      plotOptions: {
        pie: {
          dataLabels: {
            enabled: false,
            format: '{point.y:.1f}%'
          },
          tooltip: {
            valueSuffix: '$'
          }
        }
      },
      series: [
        {
          type: "pie",
          name: "Cars",
          data: data
        }
      ]
    }
  }
  showTotalSpentTime() {
    const data: [string, number][] = [];
    this.cars.forEach(car => {
      data.push([car.brand + " " + car.model,
this.countTotalSpentTimeForCar(car)]);
    });
    this._totalSpentTime = {
      chart: {
        type: "pie",
        backgroundColor: '#F3EEFE',
        borderRadius: 12,
        styledMode: false
      },
      title: {
        text: ""
      },
      xAxis: {
        categories: data.map(value => value[0])
      },
      yAxis: {
        title: {
          text: ""
        }
      },
      legend: {
        enabled: false
      },
      plotOptions: {
        pie: {
```

```
          dataLabels: {
            enabled: false,
            format: '{point.y:.1f}%'
          },
          tooltip: {
            format: '{point.y} min',
            valueSuffix: 'min'
          }
        }
      },
      series: [
        {
          type: "pie",
          name: "Cars",
          data: data
        }
      ]
    }
  }
  showCategoryChart() {
    const data: [string, number][] = [];
    if (this._selectedCar != undefined) {
      this.categories.forEach(category => {
        data.push([category.name,
this.logService.getLogsByCarAndCategory(this._selectedCar!,
category.name).length])
      })
    } else {
      this.categories.forEach(category => {
        data.push([category.name,
this.logService.getLogsByCategory(category.name).length])
      })
    }
    this._categoryChart = {
      chart: {
        type: "column",
        backgroundColor: '#FFFFFF',
        borderRadius: 12,
        styledMode: false
      },
      title: {
        text: ""
      },
      xAxis: {
        categories: data.map(value => value[0])
      },
      yAxis: {
        title: {
          text: "Amount of repairs"
        }
      },
      legend: {
        enabled: false
      },
      plotOptions: {
        pie: {
          dataLabels: {
            enabled: false,
            format: '{point.y:.1f}%'
          },
          tooltip: {
            format: '{point.y:.1f}%',
            valueSuffix: '%'
          }
```

```
        }
      },
      series: [
        {
          type: "column",
          name: "Cars",
          data: data
        }
      ]
    }
  }
}
import {Component} from '@angular/core';
import {CarService} from "../../services/car.service";
import {Car} from "../../entities/car";
import {LogService} from "../../services/log.service";
import {Log} from "../../entities/log";
import {MatDialog} from "@angular/material/dialog";
interface Category {
  name: string;
  isOpen: boolean;
}
export class CarsComponent {
  public readonly categories: Category[] = [
    {name: "Routine Maintenance (RM)", isOpen: true},
    {name: "Engine Repair", isOpen: true},
    {name: "Brake System", isOpen: true},
    {name: "Suspension and Transmission", isOpen: true},
    {name: "Electrical Equipment and Electronics", isOpen: true},
    {name: "Bodywork", isOpen: true},
    {name: "Cooling System", isOpen: true},
    {name: "Air Conditioning and Heating System", isOpen: true},
    {name: "Tuning and Modifications", isOpen: true},
    {name: "Other Tasks", isOpen: true}
  ]
  private _logsSortType = 'Sort by';
  private _carsSortType = 'Sort by';
  private _selectedCar: Car | undefined = undefined;
  private _logsForSelectedCar: Log[] = [];
  _carPreviews: Map<BigInt, string> = new Map();
  cars: Car[] = [];
  get logsSortType(): string {
    return this._logsSortType
  }
  set logsSortType(value: string) {
    this._logsSortType = value;
  }
  get carsSortType(): string {
    return this._carsSortType
  }
  set carsSortType(value: string) {
    this._carsSortType = value;
  }
  get selectedCar(): Car {
    return this._selectedCar!;
  }
  set selectedCar(car: Car | undefined) {
    this._selectedCar = car;
  }
  constructor(private dialog: MatDialog, private carService: CarService,
private logService: LogService,
              private userService: UserService, private
notificationService: NotificationService) {
    this.loadCars();}
```

```
  loadCars() {
    this.carService.getCars().subscribe(
        cars => {
          this.cars = cars;
          this.cars.forEach(car => {
            const imageUrl = '' + car.image;
            this._carPreviews.set(car.id, imageUrl);
          });
        },
        error => {
          console.error('Error loading cars:', error);
        }
    );
  }
  onAddLogClick() {
    this.dialog.open(AddLogDialogComponent, {
      data: {
        message: 'Create Log',
        buttonText: {
          ok: 'Create',
          cancel: 'Cancel'
        }
      }
    }).afterClosed().subscribe(result => {
      if (result) {
        result.carId = this.selectedCar.id;
        result.createdBy = this.userService.user.email;
        result.initiatedBy = this.userService.user.name + " " +
this.userService.user.surname;
        result.performer = undefined;
        result.isApproved = false;
        this.logService.save(result);
        this.selectCar(this.selectedCar);
        if (result.nextMaintenanceDate) {
          this.notificationService.save(new NotificationDto(
            this.selectedCar.id,
            this.userService.user.email,
            new Date(result.nextMaintenanceDate),
            undefined,
            result.category,
            result.maintenance,
            result.details,
            'ACTIVE'
          ))
        }
      }
    });
  }
  onAddCarClick() {
    this.dialog.open(AddCarDialogComponent, {
      data: {
        message: 'Create Car',
        buttonText: {
          ok: 'Create',
          cancel: 'Cancel'
        }
      }
    }).afterClosed().subscribe(result => {
      this.carService.saveCar(result).then(() => {
          this.loadCars();
      });
    });
  }
```

```
  selectCar(car: Car) {
    this._selectedCar = car;
    if (car == undefined) {
      this._logsForSelectedCar = [];
    } else {
      this._logsForSelectedCar = this.getLogsForCar(car);
    }
  }
  getLogsForCar(car: Car): Log[] {
    return this.logService.getLogsByCar(car);
  }
  updateLogsSortType() {
    if (this._logsSortType === 'MILEAGE') {
      this._logsForSelectedCar.sort((a, b) => a.mileage - b.mileage);
    } else if (this._logsSortType === 'PRICE') {
      this._logsForSelectedCar.sort((a, b) => a.price - b.price);
    } else if (this._logsSortType === 'DATE') {
      this._logsForSelectedCar.sort((a, b) => new Date(a.date).getTime() -
new Date(b.date).getTime());
    } else if (this._logsSortType === 'APPROVED') {
      this._logsForSelectedCar.sort((a, b) => {
        if (a.isApproved && b.isApproved) {
          return 0;
        } else if (a.isApproved && !b.isApproved) {
          return -1;
        }
        return 1;
      });
    }
  }
  updateCarsSortType() {
    if (this._carsSortType === 'NAME') {
      this.cars.sort((a, b) => {
        const nameA = a.brand.toUpperCase();
        const nameB = b.brand.toUpperCase();
        if (nameA < nameB) {
          return -1;
        }
        if (nameA > nameB) {
          return 1;
        }
        return 0;
      });
    } else if (this._carsSortType === 'MILEAGE') {
      this.cars.sort((a, b) => a.mileage - b.mileage);
    } else if (this._carsSortType === 'LOGS') {
      this.cars.sort((a, b) => this.logService.getLogsByCar(b).length -
this.logService.getLogsByCar(a).length);
    }
  }
  logsForCategory(category: string): Log[] {
    return this._logsForSelectedCar.filter(log => log.category ==
category);
  }
  deleteLog(log: Log) {
    this.dialog.open(ConfirmationDialogComponent,{
      data:{
        message: 'Are you sure want to delete?',
        buttonText: {
          ok: 'Delete',
          cancel: 'Cancel'
        }
      }
    }).afterClosed().subscribe(result => {
```

```
      if (result) {
        this.logService.delete(log);
        this.selectCar(this.selectedCar);
      }
    });
  }
  downloadDocs(log: Log) {
    log.docs.forEach(doc => {
      const data = window.URL.createObjectURL(doc);
      const link = document.createElement("a");
      link.href = data;
      link.download = doc.name;
      link.click();
    })
  }
  convertTimestampToDate(timestamp?: number): string {
    const date = new Date(timestamp!);
    return `${date.getFullYear()}-${(date.getMonth() +
1).toString().padStart(2, '0')}-${date.getDate().toString().padStart(2,
'0')}`;
```

Додаток З

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП ’ЮТЕРНИХ СИСТЕМ

ВЕБЗАСТОСУНОК ДЛЯ ВЕДЕННЯ ОБЛІКУ ТЕХНІЧНОГО СТАНУ ТРАНСПОРТНИХ ЗАСОБІВ

Виконав: Довженко Роман Олександрович

Керівник: доц. кафедри ПЗКС, к.т.н., доц. Заболотня Тетяна Миколаївна

Київ – 2024

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: розробити програмне забезпечення для ведення історії технічного обслуговування автомобіля.

Завдання:

1. Проаналізувати існуючі на ринку рішення.
2. Обрати засоби розроблення вебзастосунку.
3. Визначити проблеми користувача та цілі розроблення застосунку.
4. Визначити функціональні та нефункціональні вимоги до ПЗ.
5. Розробити програмне забезпечення для ведення технічного обліку транспортних засобів.
6. Протестувати застосунок на відповідність вимогам.
7. Визначити напрямки подальшого вдосконалення вебзастосунку.

АКТУАЛЬНІСТЬ

1. Необхідність у цифровому аналогу фізичним журналам ведення історії технічного обслуговування.
2. Підвищення безпеки власників транспортних засобів.
3. Необхідність зберігати документи про виконання роботи.
4. Підвищення довіри між власниками транспортних засобів та потенційними покупцями.

ІСНУЮЧІ РІШЕННЯ

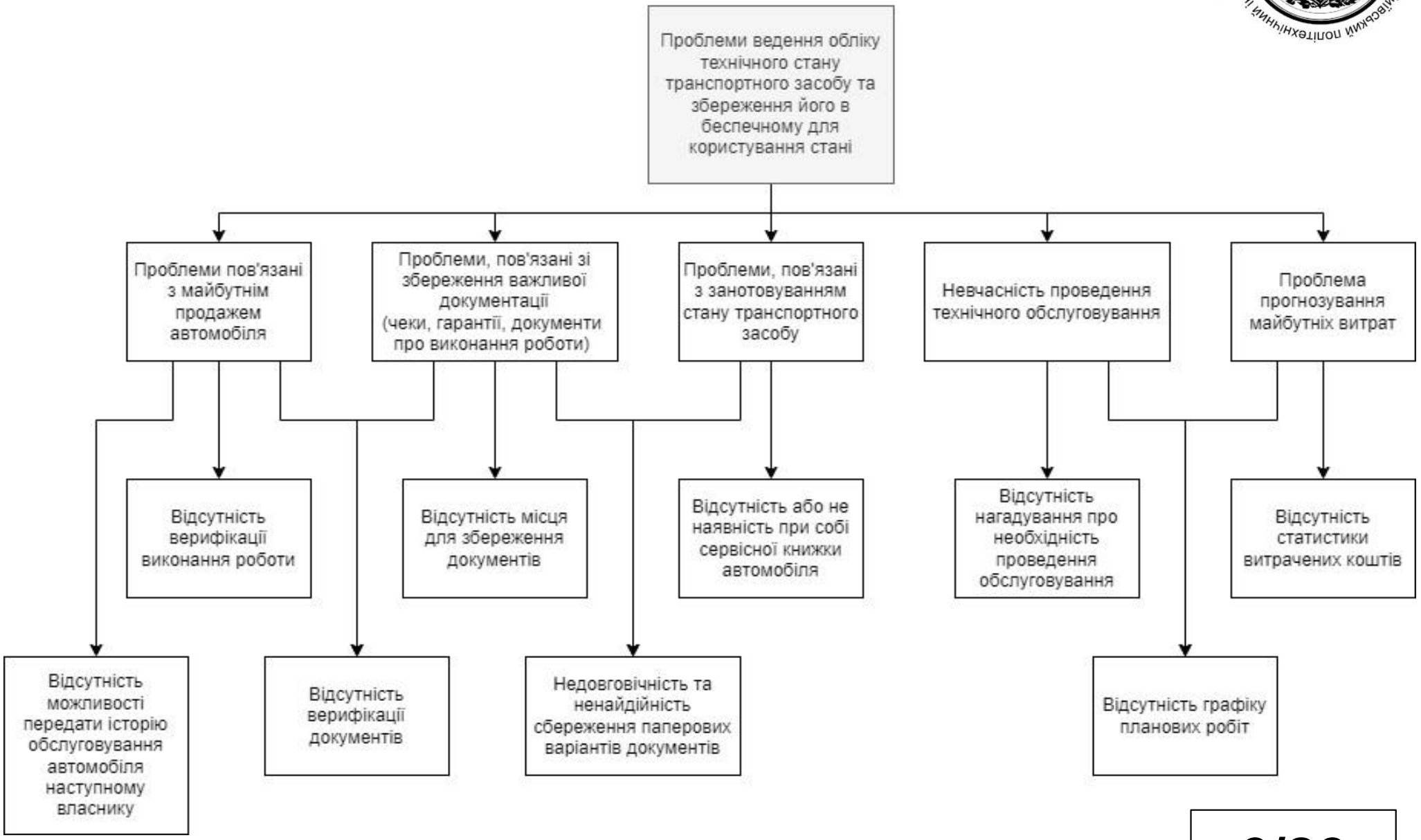
SimplyAuto

CarDiary

myCarLog

ЗАСОБИ РЕАЛІЗАЦІЇ

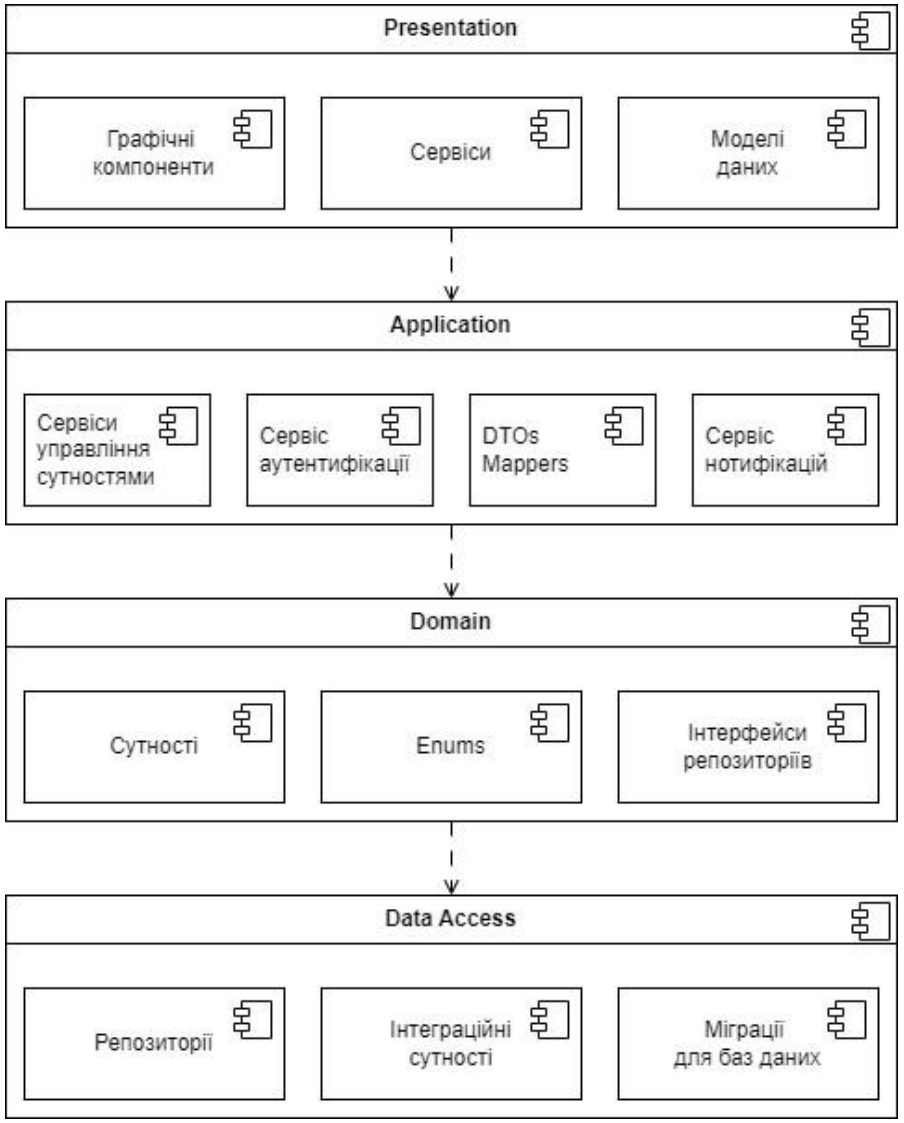
ДЕРЕВО ПРОБЛЕМ



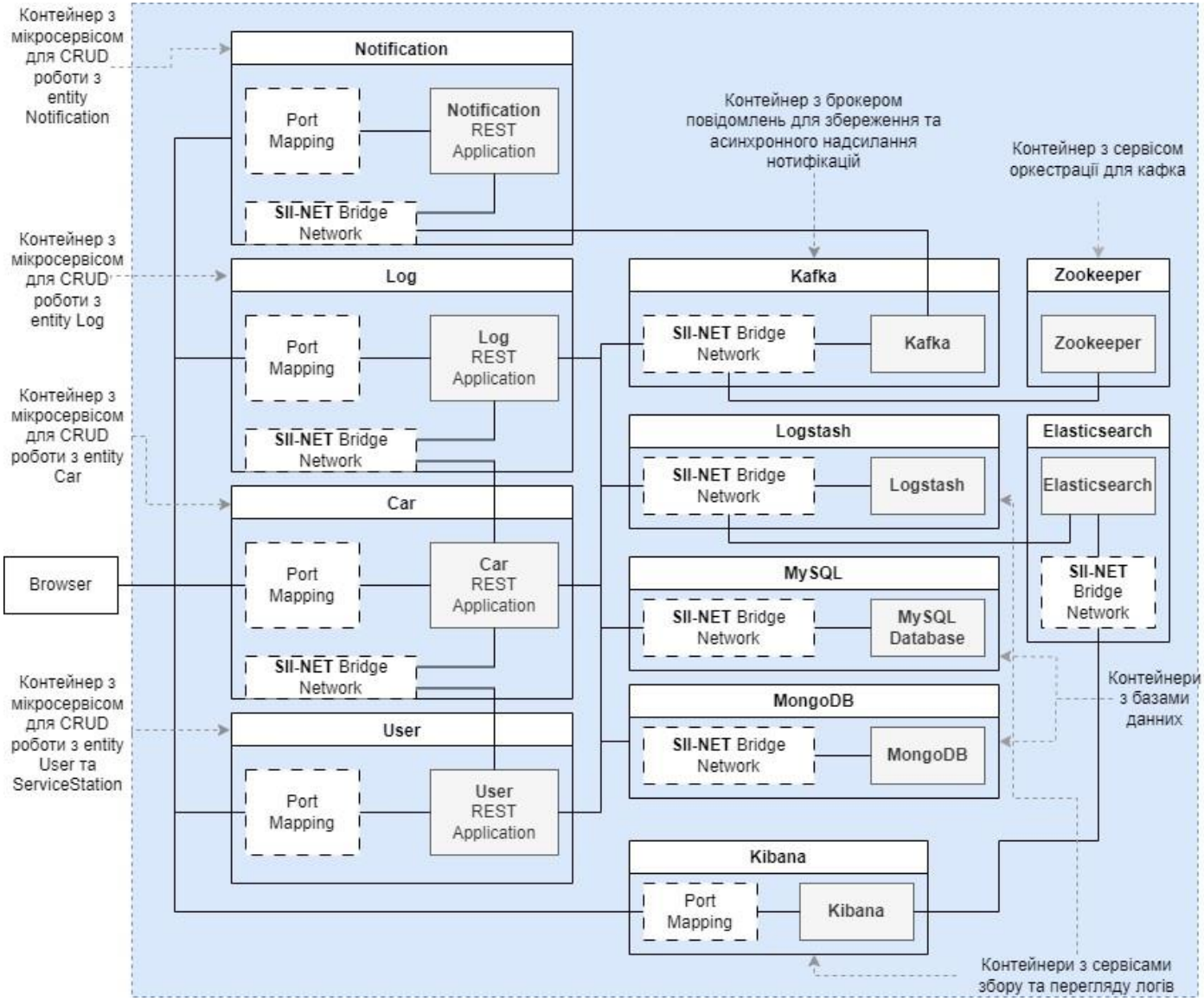
ДЕРЕВО ЦІЛЕЙ

СТРУКТУРНА СХЕМА ВЕБЗАСТОСУНКУ

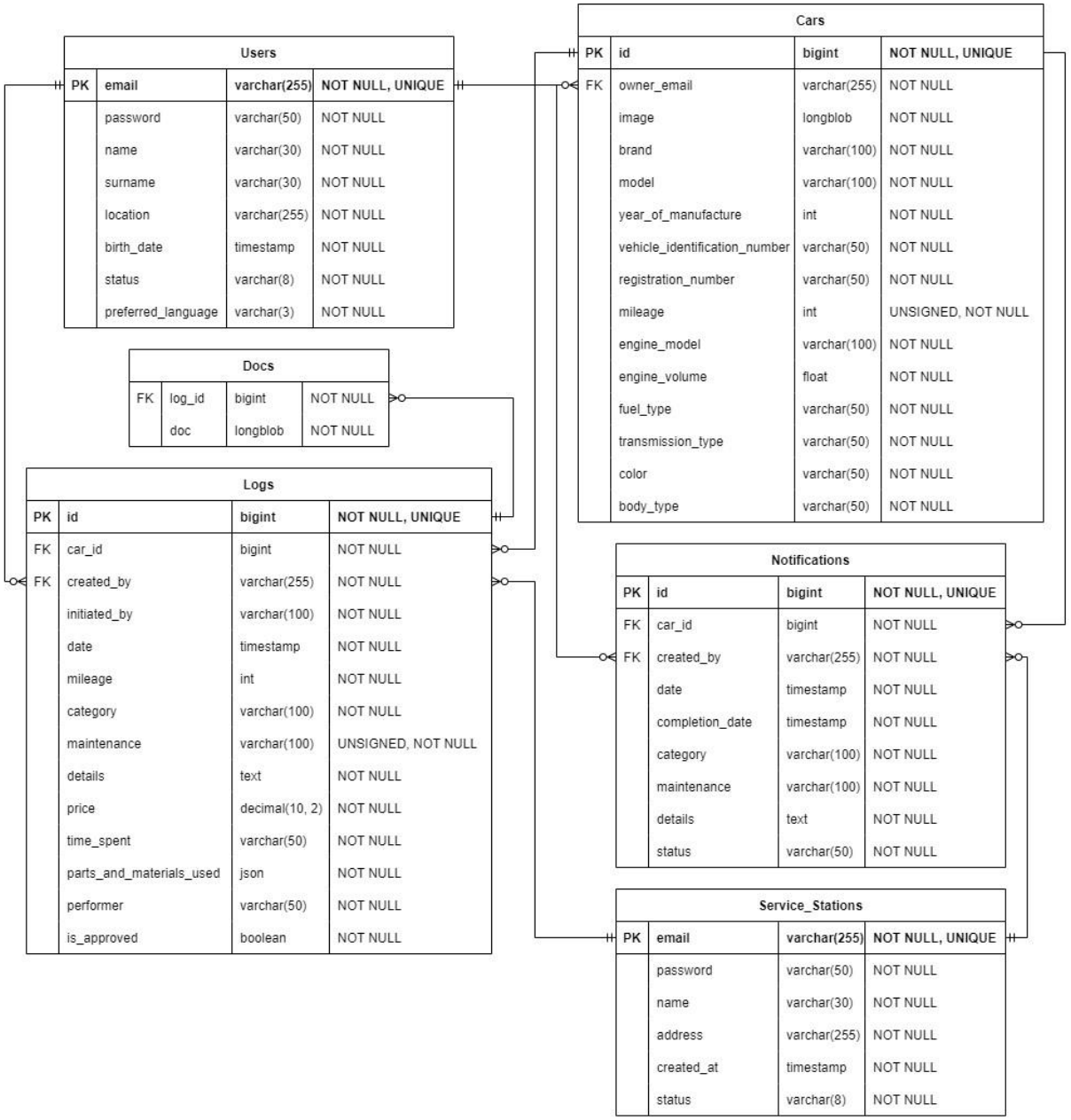
ДІАГРАМА ДЕКОМПОЗИЦІЇ ЗАСТОСУНКУ НА ЛОГІЧНІ ШАРИ



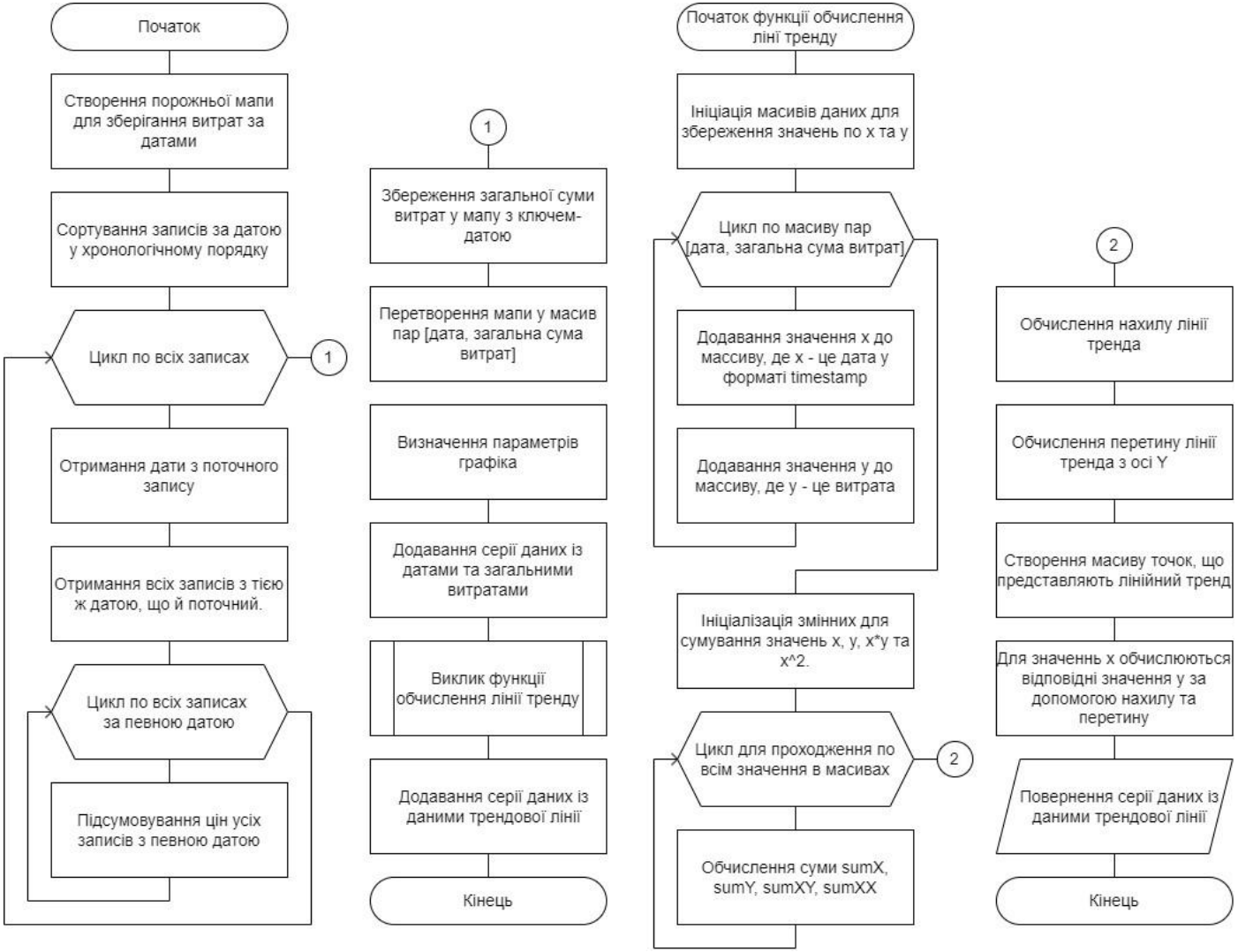
СТРУКТУРНА СХЕМА КОНТЕЙНЕРІВ СЕРВЕРНОЇ ЧАСТИНИ ВЕБЗАСТОСУНКУ



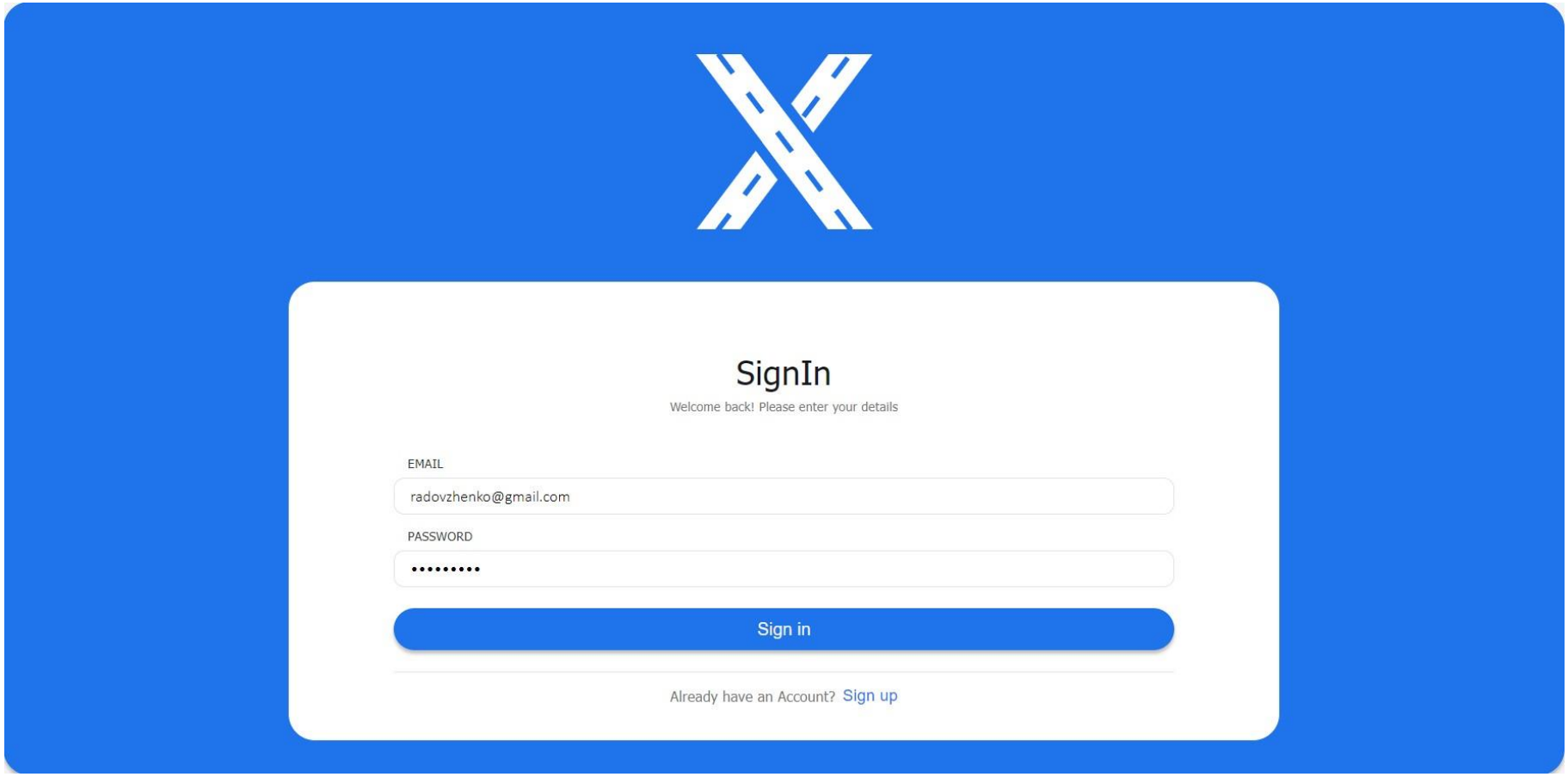
СТРУКТУРНА СХЕМА БАЗИ ДАНИХ



БЛОК-СХЕМА АЛГОРИТМУ ОБЧИСЛЕННЯ ГРАФІКУ ВИТРАТ З ЛІНІЄЮ ТРЕНДУ

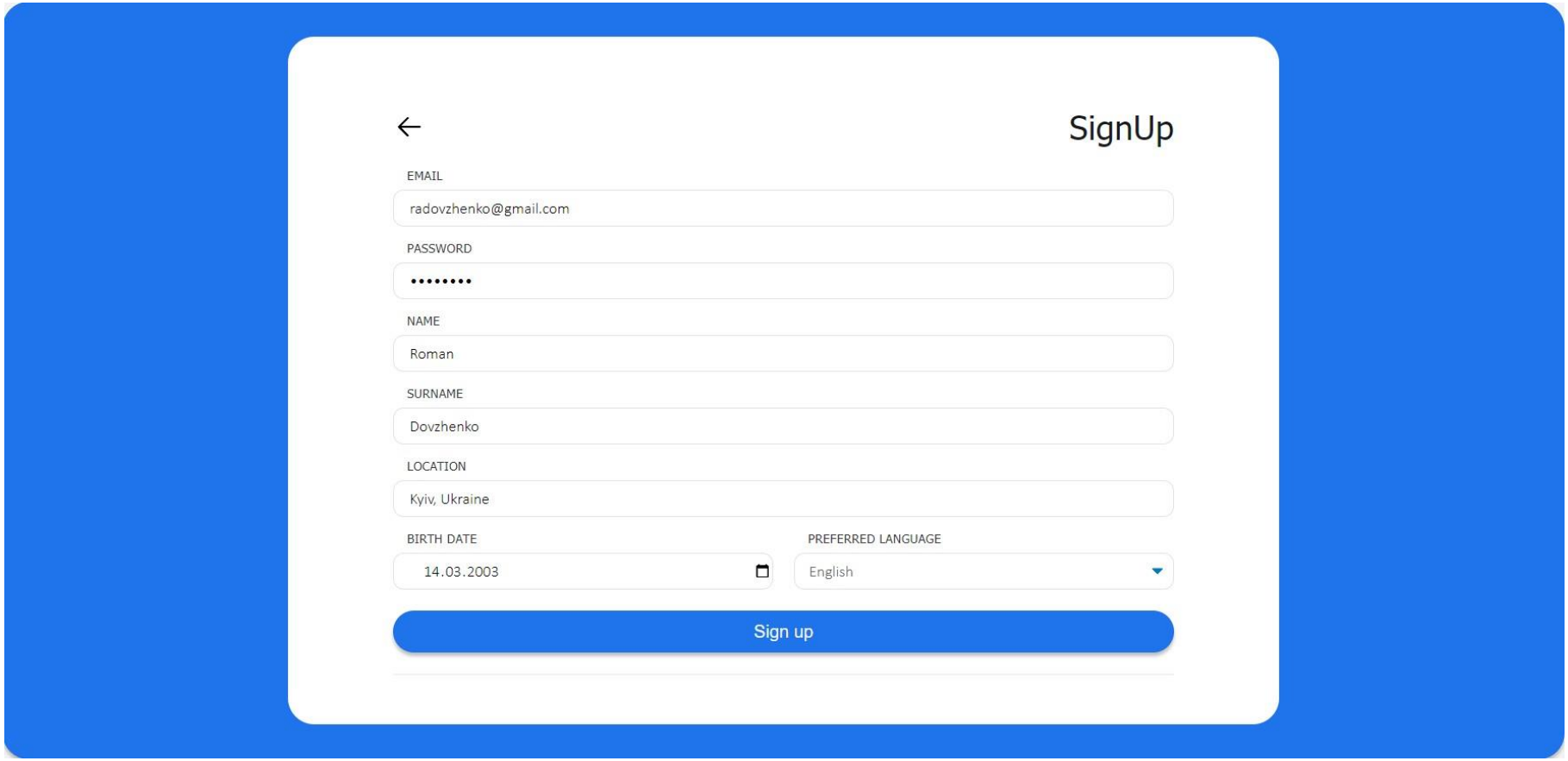


ІНТЕРФЕЙС КОРИСТУВАЧА



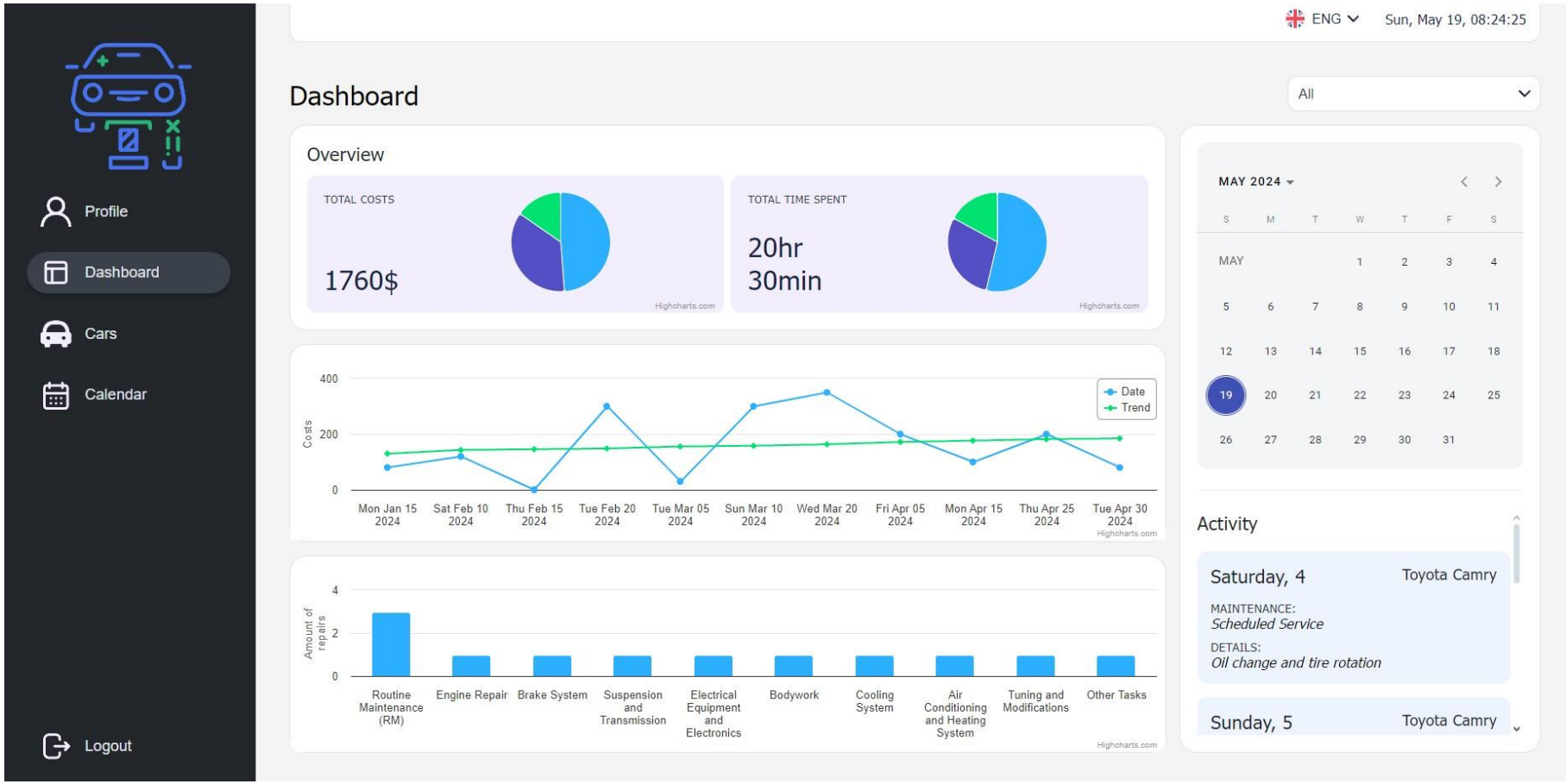
Сторінка авторизації

ІНТЕРФЕЙС КОРИСТУВАЧА



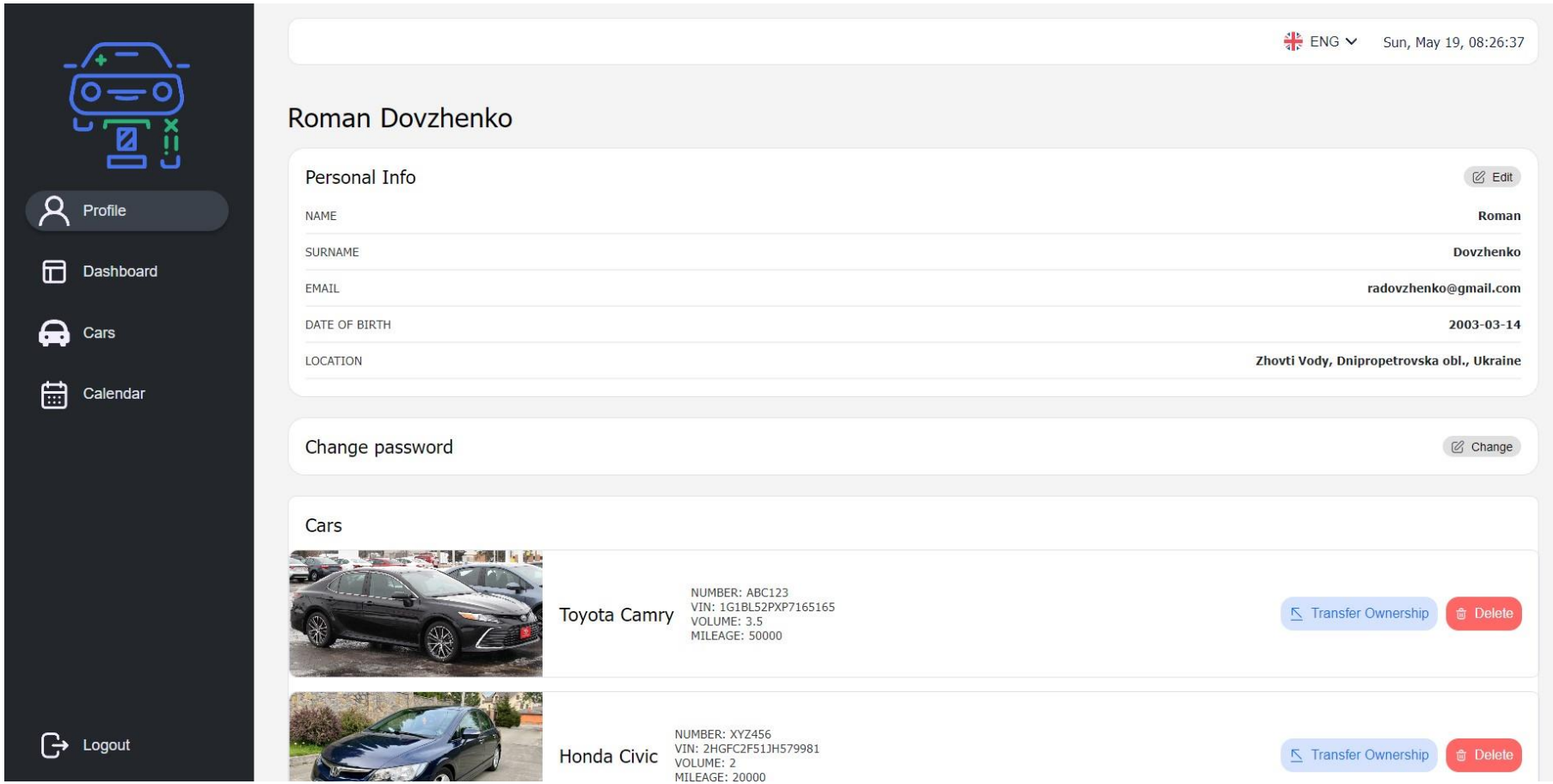
Сторінка реєстрації

ІНТЕРФЕЙС КОРИСТУВАЧА



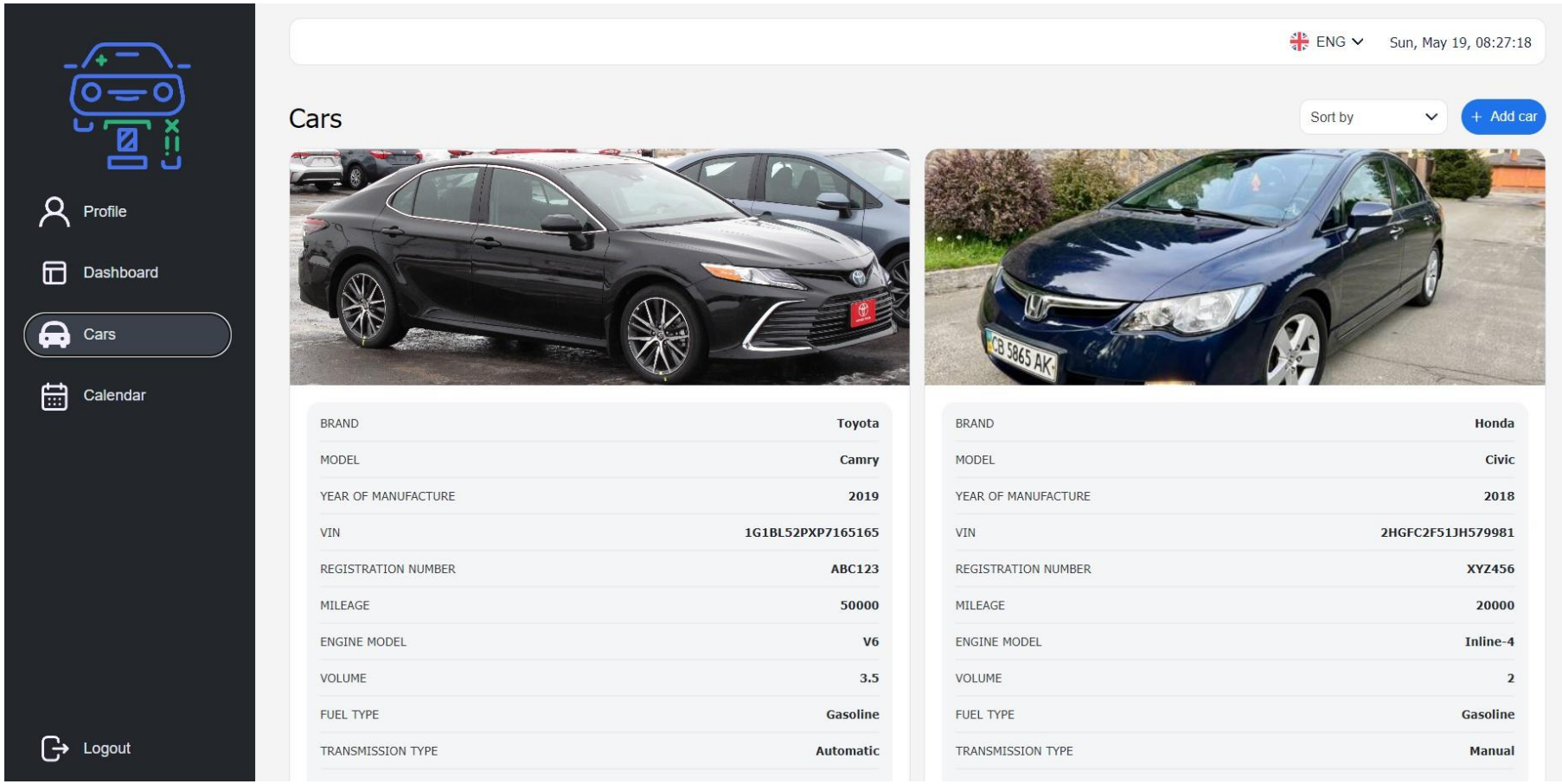
Панель моніторингу

ІНТЕРФЕЙС КОРИСТУВАЧА



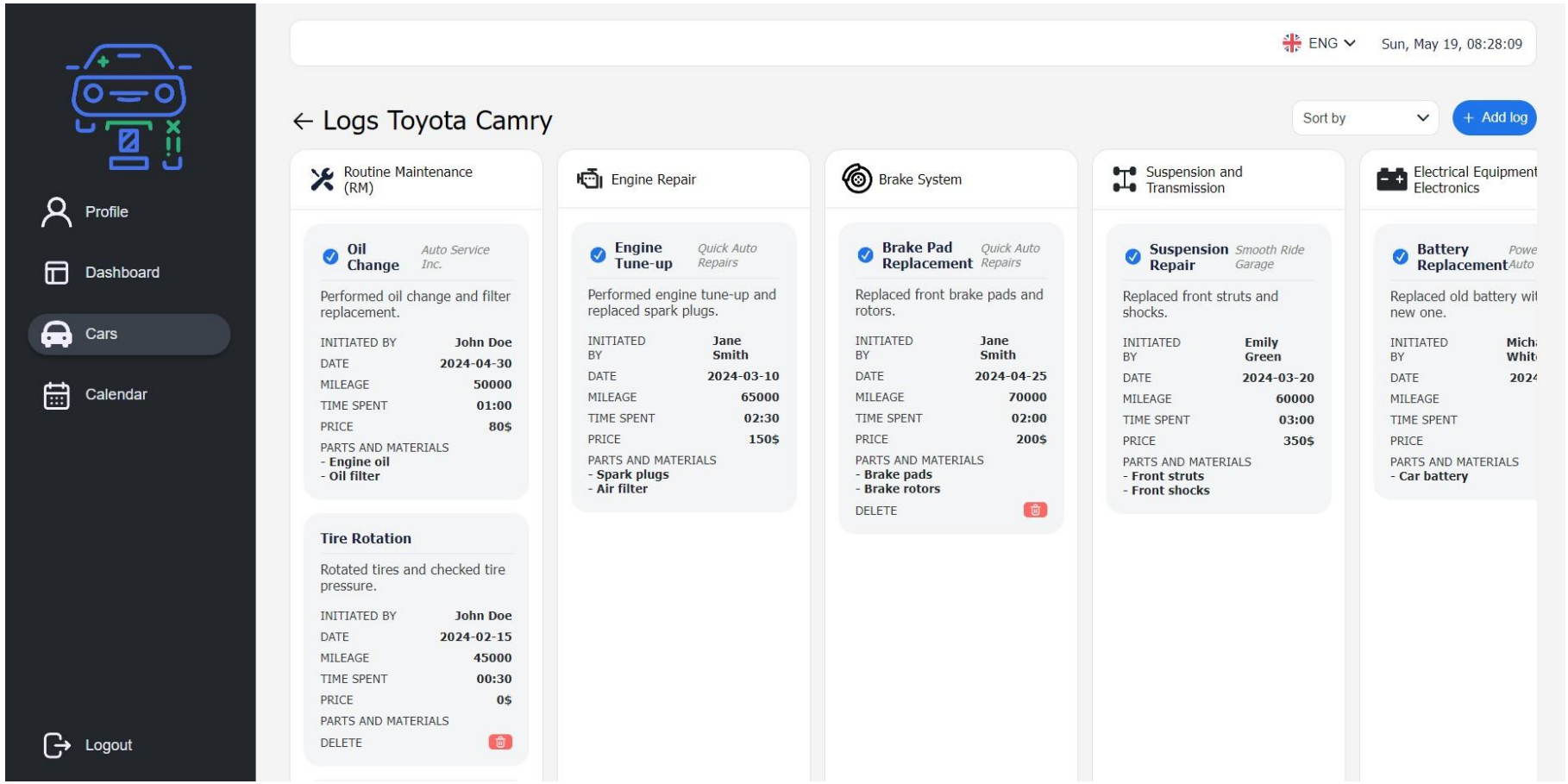
Профіль користувача

ІНТЕРФЕЙС КОРИСТУВАЧА



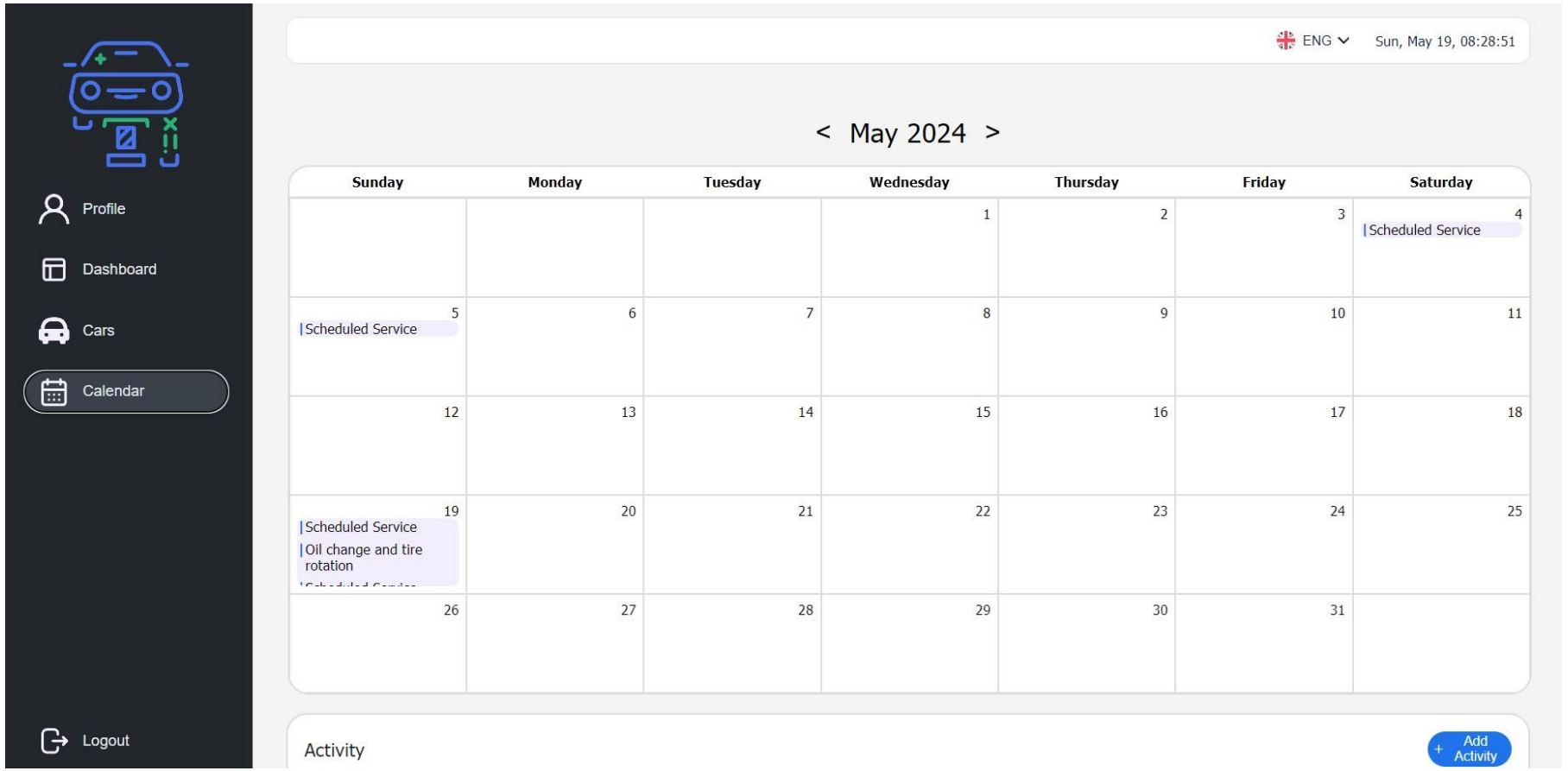
Сторінка перегляду автомобілів

ІНТЕРФЕЙС КОРИСТУВАЧА



Сторінка перегляду записів технічного обслуговування

ІНТЕРФЕЙС КОРИСТУВАЧА



Сторінка календаря

НАПРЯМКИ ПОДАЛЬШОГО ВДОСКОНАЛЕННЯ ВЕБЗАСТОСУНКУ

1. Додавання можливості автоматичного генерування звітів у форматах PDF або XLSX.
2. Можливість станціям технічного обслуговування переглядати статистику.
3. Можливість отримання сповіщень в популярних месенджерах, таких як Telegram, Viber, WhatsApp або SMS.
4. Інтеграція з іншими календарями, такими як Google Calendar та Microsoft Outlook.
5. Додавання підтримки більшої кількості мов інтерфейсу.
6. Можливість зміни теми оформлення інтерфейсу.
7. Додавання мапи найближчих сервісних станції.

ТЕСТУВАННЯ РОЗРОБЛЕНОГО ВЕБЗАСТОСУНКУ

1. Розроблено тестові сценарії для тестування основних функцій вебзастосунку.
2. Виявлено та виправлено помилки у роботі програмного забезпечення.

ВИСНОВКИ

1. Проаналізовано існуючі програмні рішення.
2. Визначено проблеми користувача, які розроблюваний застосунок має вирішувати.
3. Визначено функціональні та нефункціональні вимоги до застосунку.
4. Обрано архітектуру ПЗ та технології для програмної реалізації.
5. Визначено структуру БД.
6. Реалізовано клієнтську та серверну частини вебзастосунку відповідно вимогам.
7. Проведено тестування на відповідність вимогам.
8. Запропоновано напрямки подальшого вдосконалення вебзастосунку.

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2023 р.

ВЕБЗАСТОСУНОК ДЛЯ ВЕДЕННЯ ОБЛІКУ ТЕХНІЧНОГО СТАНУ ТРАНСПОРТНИХ ЗАСОБІВ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____Роман ДОВЖЕНКО

2023

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Вебзастосунок для ведення технічного обліку транспортних засобів, реалізований у вигляді клієнтської частини, створеної з використанням мови програмування TypeScript та фреймворку Angular, і серверної частини на Java.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) функціональна працездатність елементів сторінок вебзастосунку;
2) можливість зберігати записи про технічне обслуговування;
3) генерація статистики витрат;
4) можливість передачі історії ведення технічного обслуговування наступному власнику;
5) можливість зберігати документи виконання робіт;
6) отримання нагадувань про необхідність проведення технічного обслуговування;
7) можливість додавання, завершення та видалення запланованих подій.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Smoke Testing, що розглядається як перед випускна швидка перевірка розроблюваної системи на відповідність функціональним вимогам, а також методом White Box Testing, що включає аналіз внутрішньої структури та логіки роботи системи. Під час тестування було перевірено взаємодію між компонентами системи, основні та допоміжні функціональні можливості, клієнт-серверну взаємодію та коректність відображення сторінок вебзастосунку.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність вебзастосунку перевіряється шляхом:

1) динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;
2) динамічного ручного тестування на відповідність функціональним вимогам;
3) тестування вебзастосунку в різних браузерах на різних платформах;
4) тестування при максимальному навантаженні;
5) тестування стабільності роботи при різних умовах;
6) тестування зручності використання;
7) тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ ВЕДЕННЯ ОБЛІКУ ТЕХНІЧНОГО СТАНУ ТРАНСПОРТНИХ ЗАСОБІВ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____Роман ДОВЖЕНКО

2024

ЗМІСТ

1. Опис структури вебзастосунку

Користувацький інтерфейс можна поділити на декілька сторінок відповідно до категорії користувачів. Сторінки доступні для звичайного користувача:

- сторінка авторизації;
- сторінка реєстрації;
- сторінка панелі моніторингу;
- сторінка профілю користувача;
- сторінка перегляду автомобілів;
- сторінка перегляду записів технічного обслуговування;
- сторінка календаря.

Для працівників СТО доступна лише одна сторінка, а саме сторінка станції.

2. Процедура авторизації

Розпочати опис користувацького інтерфейсу слід зі сторінки авторизації. Це є початковою сторінкою для користувача.

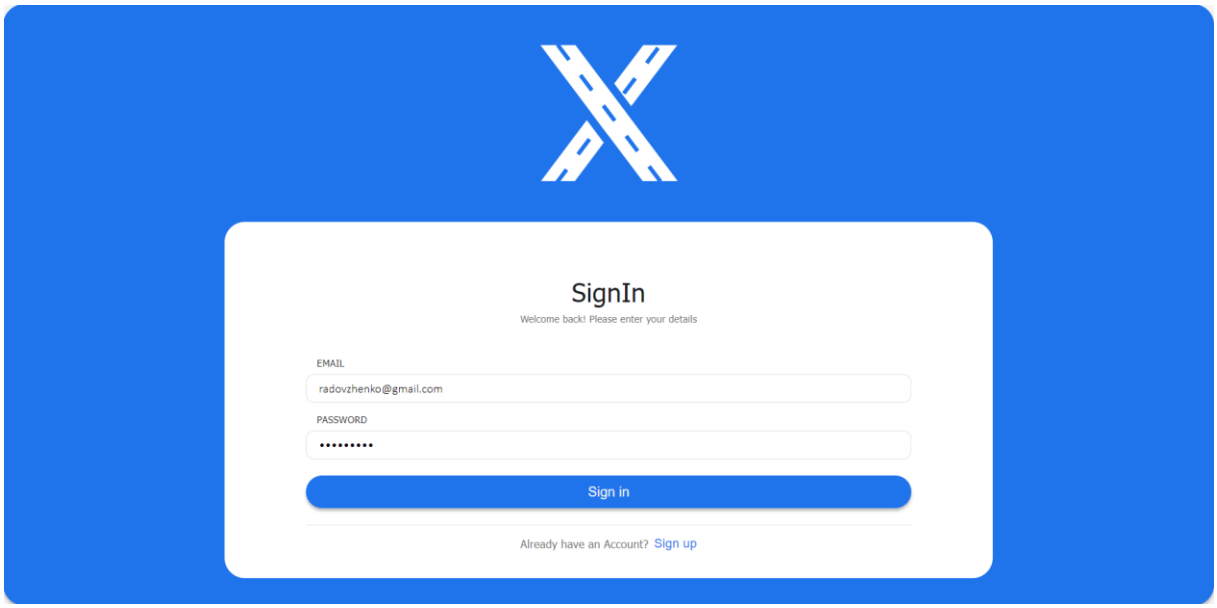


Рис. 1. Сторінка авторизації

На цій сторінці доступна можливість авторизації або переходу до сторінки реєстрації. Поля на цій сторінці підлягають валідації, і у випадку спроби авторизації з некоректним форматом електронної пошти або неправильними авторизаційними даними, користувач буде повідомлений про це.

3. Процедура реєстрації користувача

При натисканні кнопки “Sign up” на сторінці авторизації, користувач перенаправляється на сторінку реєстрації.

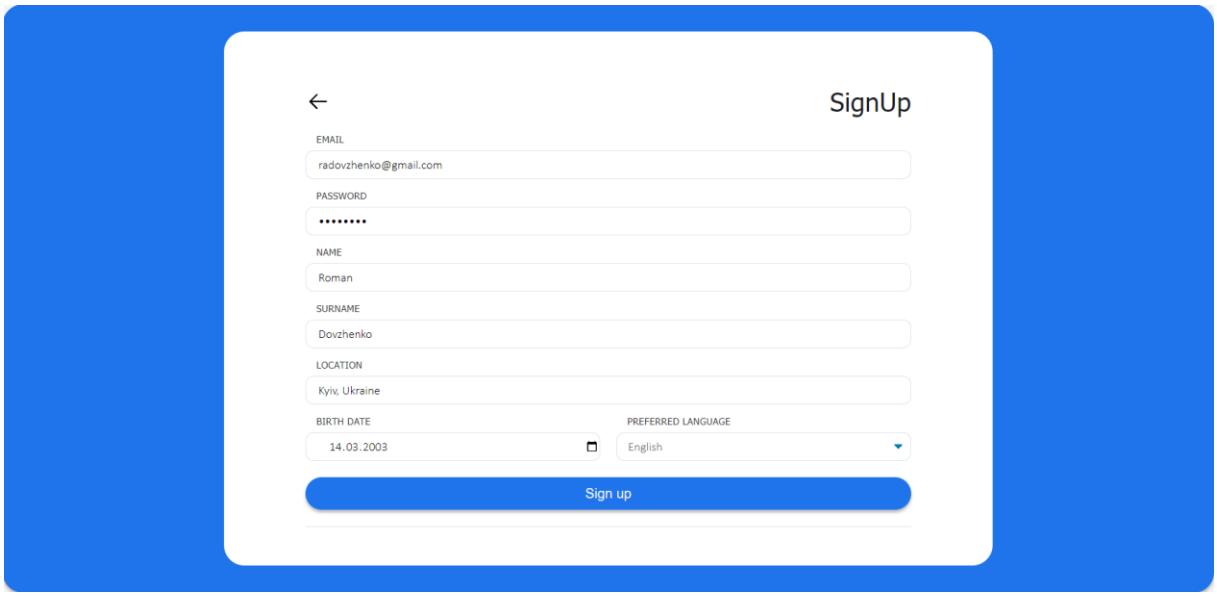


Рис. 2. Сторінка реєстрації

Сторінка реєстрації містить поля для створення нового користувача поля. Всі ці поля обов'язкові до заповнення і підлягають валідації. Користувач заблокований в завершені реєстрації у випадку незаповнення всіх полів. Поля для введення електронної пошти та паролю мають додаткову перевірку на коректність формату і безпековим вимогам.

У випадку коректного заповнення всіх полів і натисканні кнопки завершення процесу реєстрації, користувача буде переадресовано до сторінки авторизації.

4. Панель моніторингу

Попередньо зареєстрованого та успішно авторизованого користувача системи зустрічає панель моніторингу. На всіх сторінках, окрім реєстрації та авторизації користувача, доступне навігаційне меню та панель стану, на якій є можливість змінити мову інтерфейсу.

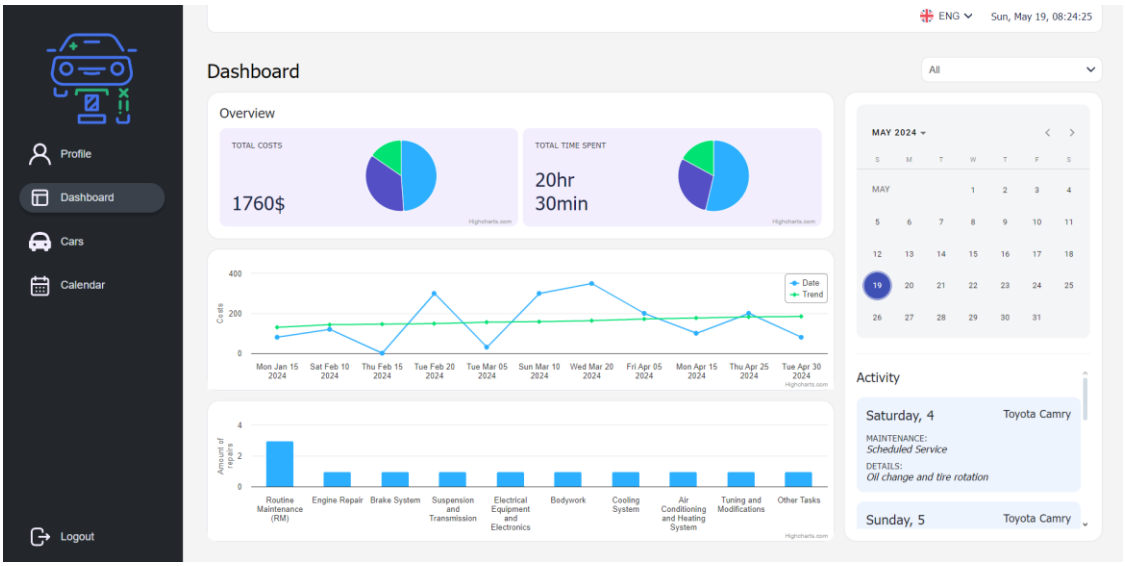


Рис. 3. Панель моніторингу

На сторінці панелі моніторингу доступна статистична інформація щодо записів проведення технічного обслуговування. Насамперед, це дві секторні діаграми, що вказують на загальні витрати грошей та часу. При наведенні на сектор однієї з цих діаграм контекстним вікном буде продемонстровано витрати за певним сектором, у даному випадку сектором є автомобіль. Нижче від секторних діаграм наведено лінійну діаграму з лінією тренду, що демонструє витрати залежно від дати. При наведенні на будь-яку точку цього графіку, контекстне вікно проінформує користувача про витрату, яка була зроблена протягом цієї дати. Закінчується блок діаграм стовпчастою діаграмою, що ілюструє статистику кількості записів за категорією.

В правій частині панелі моніторингу зображено календар та список майбутніх подій, які відбудуться у цьому місяці. Для отримання більш

детальної інформації та можливості модифікувати події користувач може перейти на сторінку календаря.

Варто зазначити, що у користувача є можливість відобразити продемонстровану інформацію лише для обраного автомобіля шляхом вибору його у розкривному списку. На навігаційному меню користувачу доступні кнопки для переходу до сторінок профілю, автомобілів, календаря та панелі моніторингу.

5. Сторінка профілю

Сторінка профілю користувача в цілому складається з трьох блоків. Перший блок – це форма з особистою інформацією. На цій формі відображена основна інформація про користувача, а також присутня кнопка, що відкриває вікно для зміни даних користувача. Другий блок фактично складається лише з однієї кнопки, яка відповідає за відкриття вікна зміни пароля. І, нарешті, третій блок – це перелік всіх автомобілів користувача. Біля кожного автомобіля розміщена невелика інформація щодо нього, а також кнопки для передачі прав власності з усією історією технічного обслуговування та видалення автомобіля.

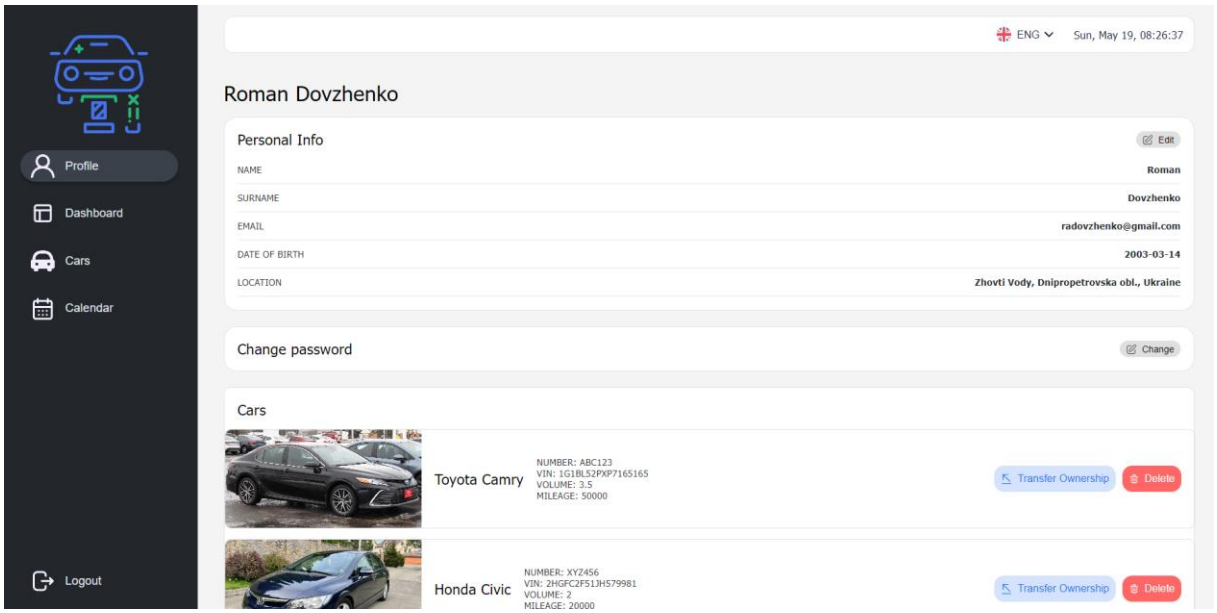


Рис. 4. Профіль користувача

6. Модифікація персональної інформації

При натисканні на кнопку модифікації персональної інформації користувача відкривається діалогове вікно, що містить поля для заповнення. Поле для введення електронної пошти заблоковане до модифікації, оскільки електронна пошта є унікальним ідентифікатором користувача в системі і не підлягає зміні. Для всіх інших полів наявна перевірка на їх заповненість. У разі незаповненості одного з полів користувач буде заблокований від завершення процесу модифікації персональних даних.

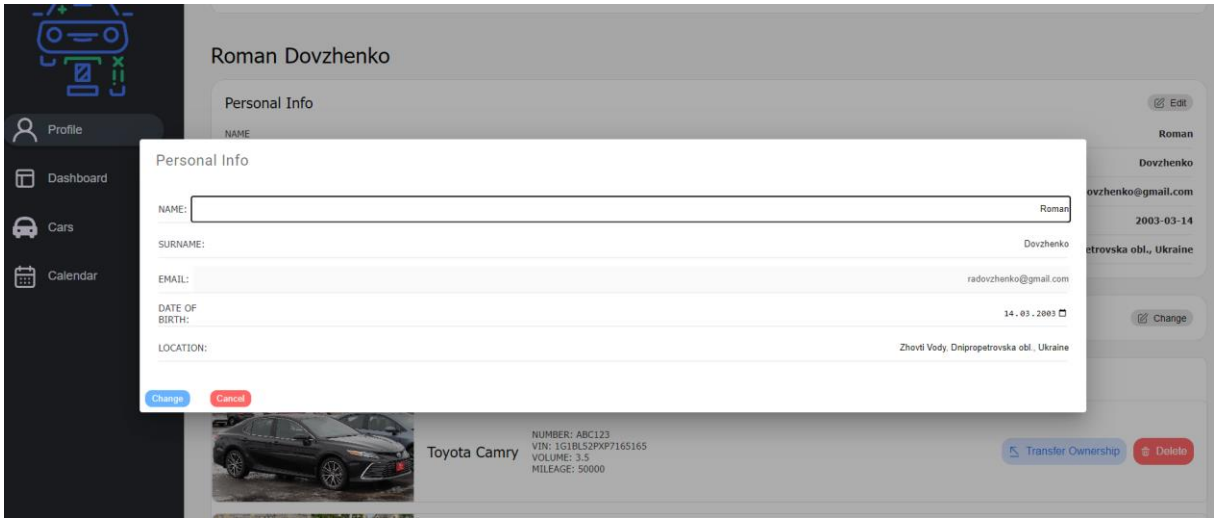


Рис. 5. Вікно модифікації даних користувача

7. Сторінка перегляду автомобілів

Сторінка перегляду автомобілів демонструє користувачеві створені ним автомобілі. Автомобілі та інформація про них представлені у вигляді карток. На сторінці доступна кнопка для додавання нового автомобіля, а також розкривний список для фільтрації існуючих автомобілів за назвою, пробігом та кількістю записів технічного обслуговування. Кожна карточка автомобіля, а саме його зображення, по суті є кнопкою для переходу до сторінки перегляду записів.

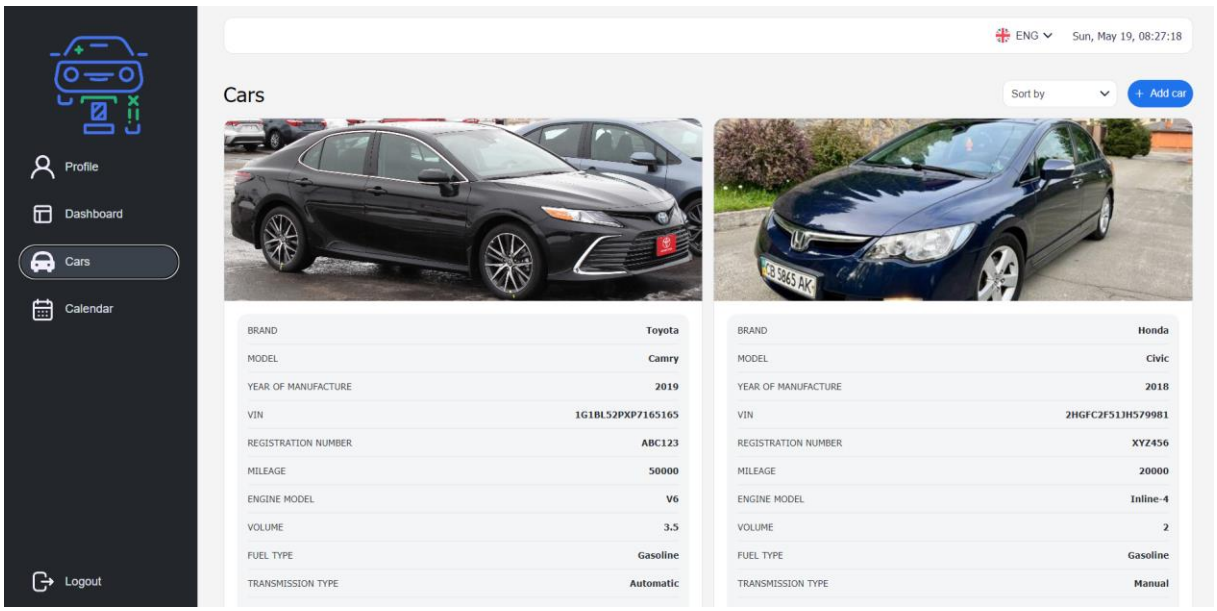


Рис. 6. Сторінка перегляду автомобілів

8. Додавання нового автомобіля

При натисканні кнопки додавання нового автомобіля користувачу відображається діалогове вікно створення автомобіля. Це діалогове вікно містить вікно для завантаження зображення автомобіля та форму для заповнення інформації щодо автомобіля.

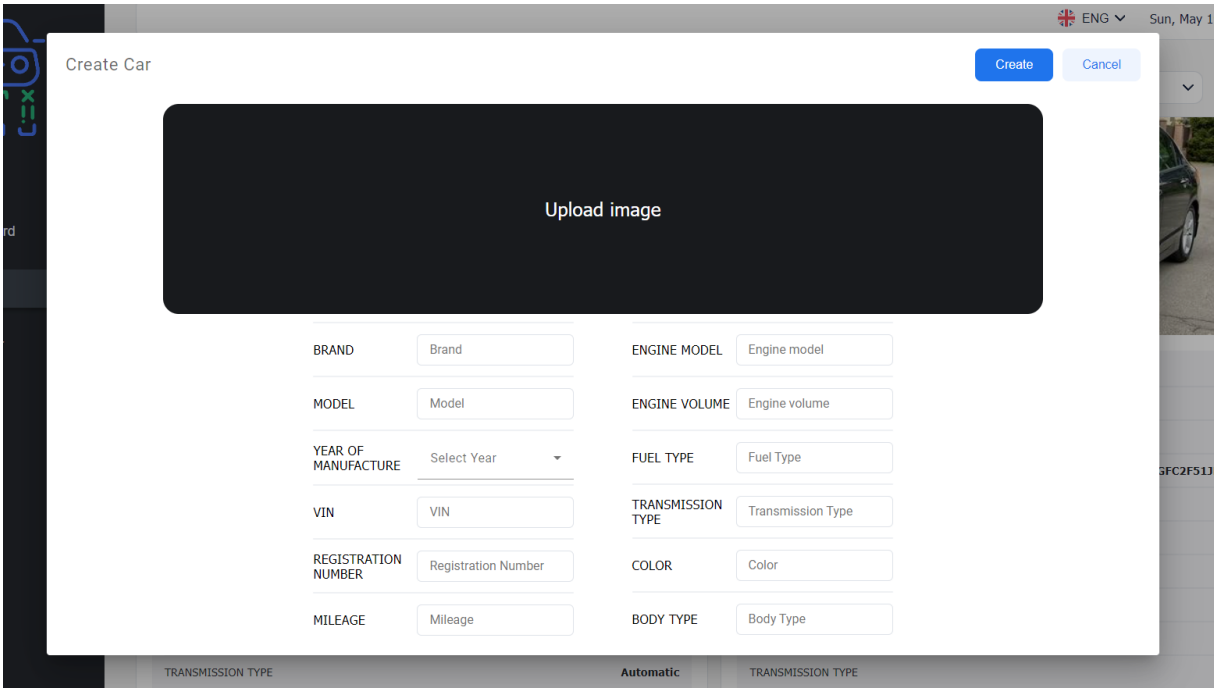


Рис. 7. Вікно додавання автомобіля

У разі успішного завантаження зображення автомобіля користувачу буде доступний попередній його перегляд. Усі поля в формі інформації є обов'язковими для заповнення, а деякі з них підлягають додатковій перевірці на невід'ємність, зокрема поля пробігу та об'єму двигуна. У разі некоректного введення інформації або незаповнення всіх полів користувач буде повідомлений про це та заблокований від подальшого завершення створення автомобіля.

9. Перегляд записів технічного обслуговування

Як уже зазначено, натискання на зображення автомобіля відкриває користувачеві сторінку перегляду записів технічного обслуговування. Ця сторінка містить всі записи про обслуговування автомобіля, об'єднані категоріями, а також кнопку для додавання нового запису та розкривний список для сортування записів за пробігом, ціною, датою та верифікацією.

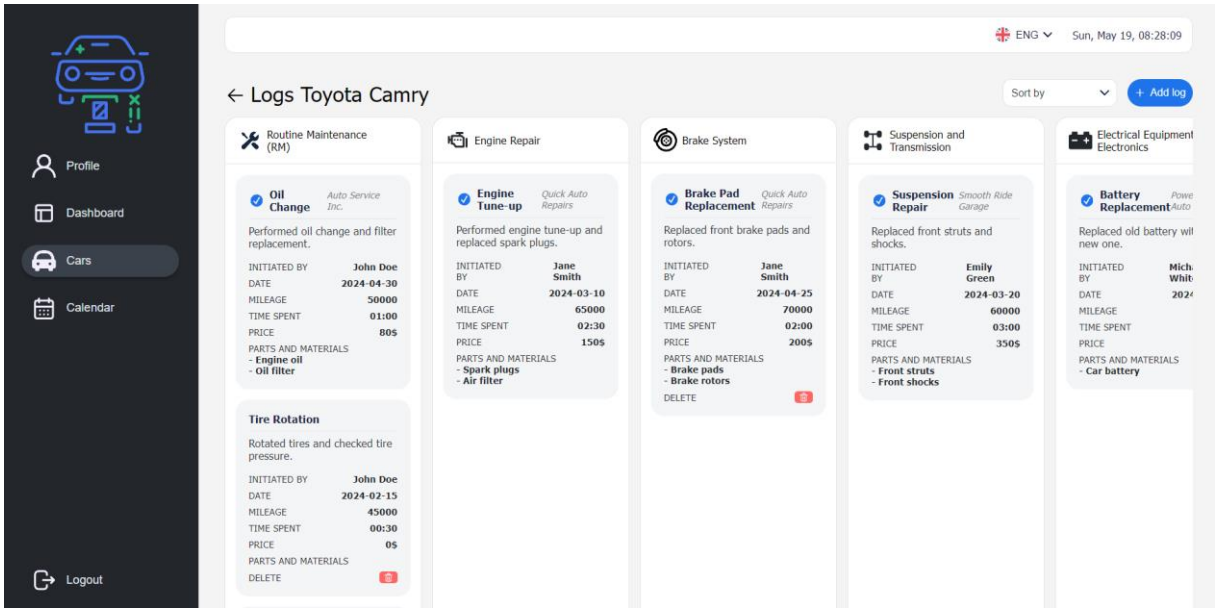


Рис. 8. Сторінка перегляду записів технічного обслуговування

Записи про проведене обслуговування представлені у вигляді карточок. Кожна карточка містить назву запису та детальну інформацію щодо того, що було проведено, скільки це коштувало, скільки на це було

витрачено часу і т.д., а також перелік матеріалів, які були використані, кнопку видалення запису та кнопку завантаження документів виконаної роботи у разі, якщо вони були додані. Для підтверджених записів біля назви запису спеціальною іконкою продемонстровано те, що вони є затвердженими станцією технічного обслуговування, а поруч написано назву станції, в якій проводилось обслуговування.

10. Додавання нового запису про технічне обслуговування

Натискання кнопки додавання нового запису відкриває користувачеві вікно додавання запису про технічне обслуговування. Це вікно складається з розкривного списку вибору категорії обслуговування, обов'язкових до заповнення полів інформації стосовно запису, необов'язкового для заповнення поля про дату наступного обслуговування, а також кнопок додавання витратних матеріалів та документів.

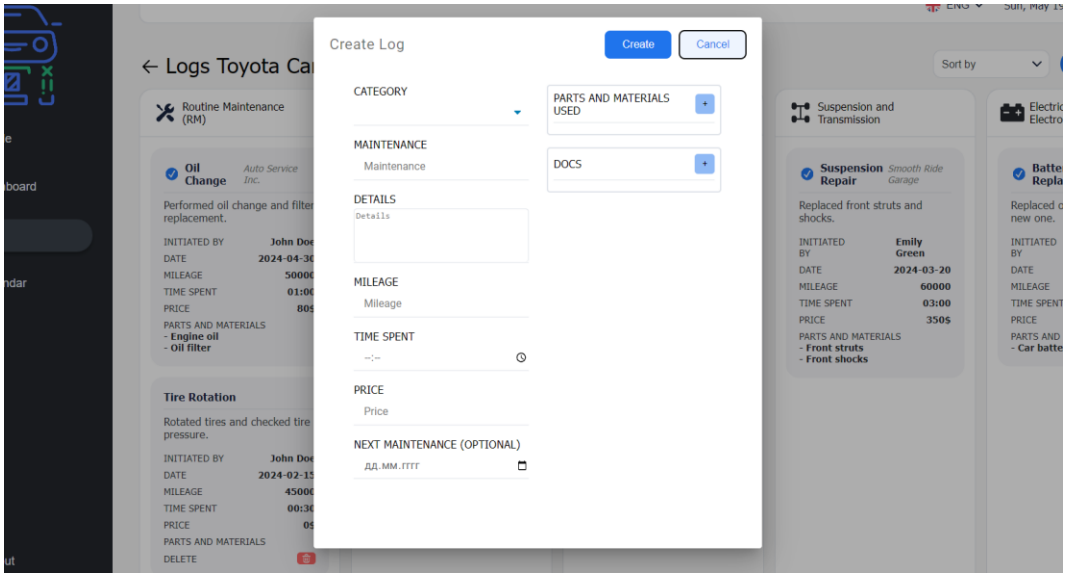


Рис. 9. Вікно додавання запису про технічне обслуговування

У разі незаповнення обов'язкових полів користувач буде повідомлений про це та заблокований в подальшому завершенні процесу створення запису. При натисканні кнопки додавання витратного матеріалу відкривається діалогове вікно з обов'язковим для заповнення полем, в

якому користувачу пропонується ввести назву матеріалу. У разі успішного додавання витратного матеріалу або документу біля нього також буде відображена кнопка видалення. Заповнення необов'язкового поля з датою наступного обслуговування, у разі збереження запису, створює відповідну подію в календарі.

11. Календар подій

Вибір розділу з календарем у навігаційному меню переадресовує користувача на відповідну сторінку. Загалом, сторінку календаря можна поділити на інтерактивний календар та перелік запланованих подій. Інтерактивний календар являє собою таблицю, де кожна комірка представляє день місяця. Зверху над календарем є стрілки для вибору місяця. Залежно від наявності подій, комірка може бути порожньою або заповненою спеціальною карткою, яка перераховує події, зазначені на цю дату.

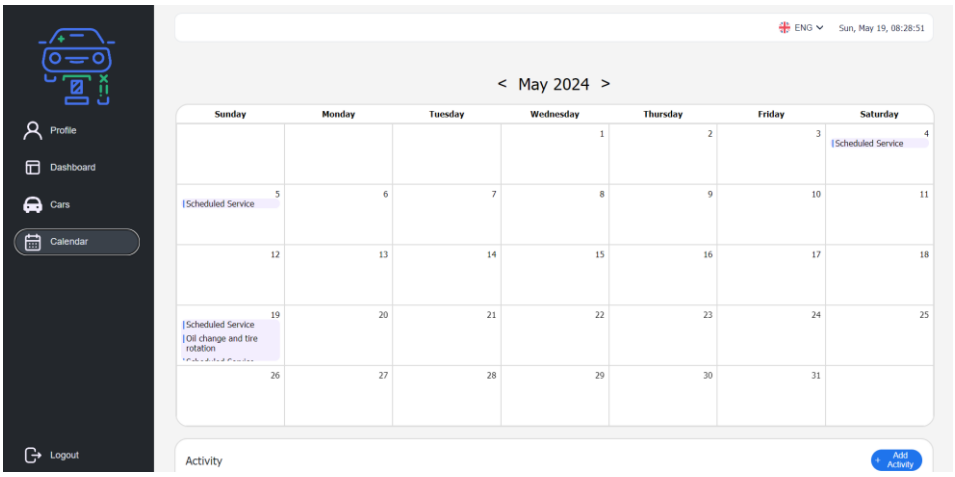


Рис. 10. Сторінка календаря

12. Список запланованих подій

Список запланованих подій, представлений нижче інтерактивного календаря, містить перелік подій, відсортованих за датою. Кожна подія представлена карткою, на якій подано детальну інформацію, а також

наявні кнопки для видалення та виконання події. Виконання події вимикає подальші нагадування. Виконана подія також відображається в календарі перекресленою. Кнопка видалення події має діалогове вікно підтвердження, і у разі підтвердження подія видаляється зі списку та календаря.

Рис. 11. Список запланованих подій на сторінці календаря

13. Додавання події

Кнопка додавання події відкриває користувачу діалогове вікно для створення нової події. Вікно містить розкривний список для вибору автомобіля, до якого буде прив'язана ця подія, розкривний список категорії обслуговування, назву операції, деталі та дату проведення.

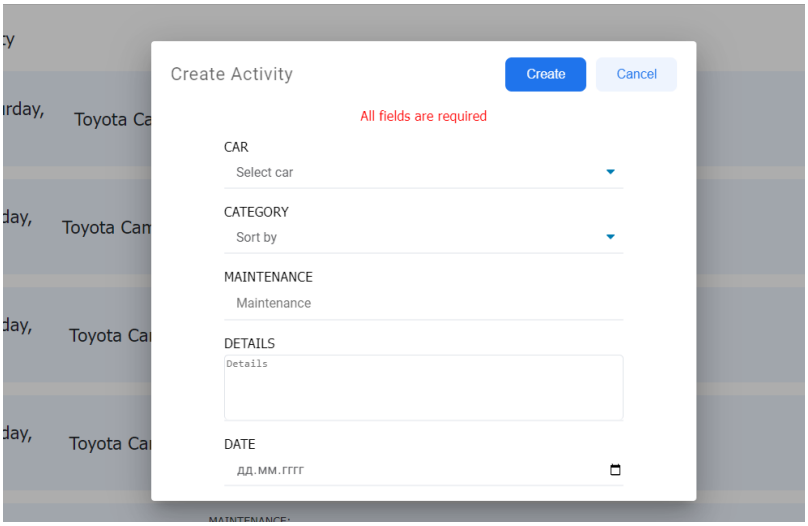


Рис. 12. Вікно додавання події

Всі ці поля обов'язкові для заповнення, і у разі незаповнення одного з них користувач буде повідомлений про це та заблокований від подальшого збереження.

14. Сторінка працівника СТО

Тепер розглянемо сторінки, доступні для авторизованого працівника СТО. Сторінка авторизації в такому випадку ідентична до сторінки для звичайного користувача.

Для працівника СТО доступна лише одна сторінка. На цій сторінці представлена панель стану та поле для вводу електронної пошти користувача, що обслуговується. Це поле містить перевірку на правильний формат, і у разі його порушення система повідомить про це.

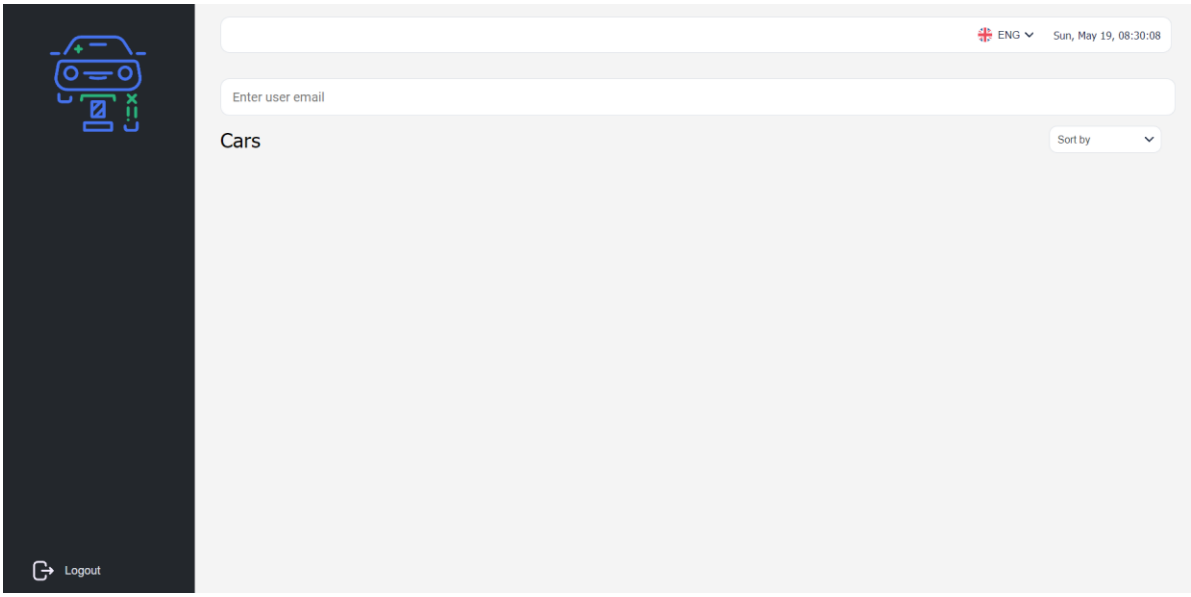


Рис. 13. Сторінка станції технічного обслуговування

15. Пошук автомобілів користувача

Введення та підтвердження електронної пошти користувача, який вже існує в системі, надає працівнику СТО доступ до переліку автомобілів. Це вікно ідентичне тому, що доступне звичайному користувачу. Натискання на зображення автомобіля переадресовує працівника СТО до сторінки перегляду записів технічного обслуговування.

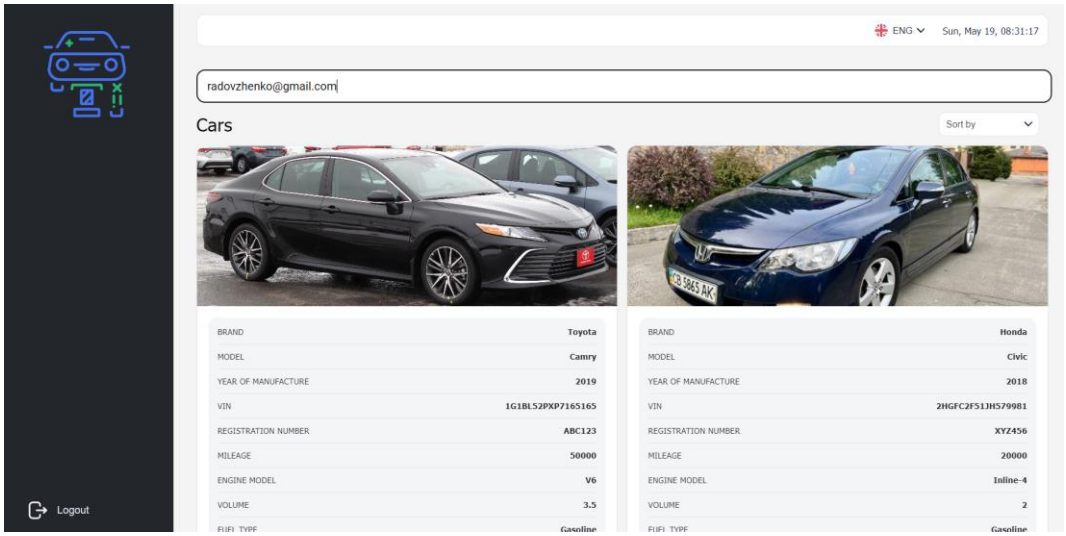


Рис. 14. Відображення автомобілів за пошуковим запитом пошти

16. Перегляд записів за пошуковим запитом

Натискання на зображення автомобіля дозволяє працівнику СТО переглянути записи про проведені технічні операції над автомобілем та, при необхідності, додати новий. Для звичайного користувача запис, доданий працівником станції технічного обслуговування, є підтвердженим і позначеним спеціальною відміткою. Вікно додавання запису для працівника СТО ідентичне вікну звичайного користувача.

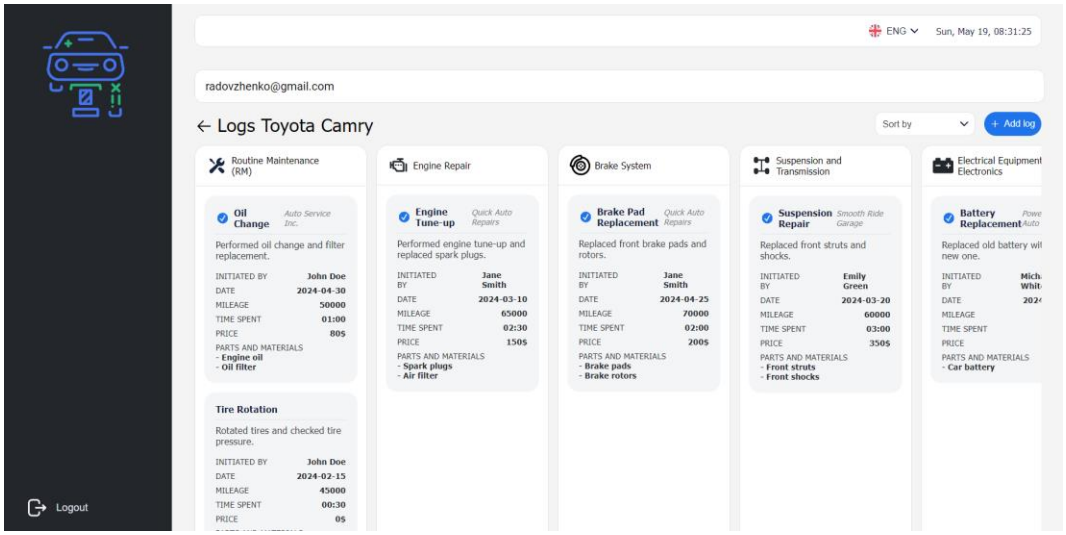


Рис. 15. Відображення записів технічного обслуговування за пошуковим запитом пошти