

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Інформаційна система підбору автомобільних запчастин»

Виконав студент групи ПЗ-22-2
ТРОФІМЧУК Микола Олександрович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
ЄФРЕМОВ Юрій Миколайович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ТРОФІМЧУК Микола Олександрович**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Інформаційна система підбору автомобільних запчастин»,**
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., № 92/с**

Вихідні дані для роботи: **інформаційні технології електронної комерції для**
розробки системи пошуку та підбору автомобільних запчастин.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	01.03.2026р.	01.03.2026р.
2	Власенко О.В.	01.03.2026р.	01.03.2026р.
3	Власенко О.В.	01.03.2026р.	01.03.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебзастосунку для підбору автомобільних запчастин та пояснювальної записки. Її текстова частина викладена на 74 сторінках. Список джерел містить 30 найменувань на 3 сторінках. У роботі наведено 38 ілюстрацій, 19 таблиць та 17 сторінок додатків. Загальний обсяг – 96 сторінок.

Головна мета роботи – розробка сучасної масштабованої інформаційної системи підбору автозапчастин, що забезпечує зручне керування товарними каталогами, швидкий повнотекстовий пошук, точну перевірку сумісності деталей з автомобілями, а також інструменти «Віртуального гаража» й інтелектуального помічника ТО.

У ході проєкту проведено системний аналіз, сформульовано вимоги та обрано архітектуру серверної частини. Спроєктовано реляційну базу даних та реалізовано алгоритми бекенду з інтеграцією зовнішніх сервісів. Клієнтську частину реалізовано як SPA-застосунок із підтримкою візуальних діаграм та рольовим розмежуванням доступу. Для стабільності платформи розгорнуто контейнеризоване середовище пошукового рушія та налаштовано конвеєр CI/CD, що підтверджено результатами третього розділу.

КЛЮЧОВІ СЛОВА: REACT, .NET 8, CLEAN ARCHITECTURE, POSTGRESQL, ELASTICSEARCH, CLOUDINARY, ENTITY FRAMEWORK CORE, JWT, REST API, ПІДБІР ЗАПЧАСТИН, СУМІСНІСТЬ АВТО, ВІРТУАЛЬНИЙ ГАРАЖ, VIN-ДЕКОДУВАННЯ.

					ІПЗ.КР.Б – 121 – 26 - ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата	Інформаційна система підбору автомобільних запчастин Пояснювальна записка	Літ.		Арк.	Аркушів
Розроб.		Трофімчук М.О.						3	90
Керівник		Власенко О.В.							
Рецензент		Єфремов Ю.М.							
Зав. каф.		Вакалюк Т.А.				Житомирська політехніка, група ІПЗ-22-2			

ABSTRACT

The bachelor's qualification thesis consists of a web application for auto parts selection and an explanatory note. Its textual part is presented on 74 pages. The list of references includes 30 sources on 3 pages. The work contains 38 illustrations, 19 tables, and 17 pages of appendices. The total volume is 96 pages.

The main goal of the work is to develop a modern, scalable information system for auto parts selection that ensures convenient management of product catalogs, fast full-text search, precise checking of part compatibility with vehicles, as well as "Virtual Garage" tools and an intelligent maintenance assistant.

During the project, a system analysis was conducted, requirements were formulated, and the server-side architecture was chosen based on Clean Architecture principles. A relational database was designed, and backend algorithms with external service integrations were implemented. The client side was developed as an SPA featuring visual diagram support and role-based access control. To ensure platform stability, a containerized search engine environment was deployed and a CI/CD pipeline was configured, which is confirmed by the results in the third chapter.

KEYWORDS: REACT, .NET 8, CLEAN ARCHITECTURE, POSTGRESQL, ELASTICSEARCH, CLOUDINARY, ENTITY FRAMEWORK CORE, JWT, REST API, AUTO PARTS SELECTION, VEHICLE COMPATIBILITY, VIRTUAL GARAGE, VIN DECODING.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД СУЧАСНИХ ІНФОРМАЦІЙНИХ СИСТЕМ	9
1.1. Постановка завдання	9
1.2. Аналіз існуючого програмного забезпечення	12
1.3. Вибір архітектури системи	16
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	20
Висновки до першого розділу	23
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ	25
2.1. Аналіз вимог та специфікація системи	25
2.2. Проєктування бази даних	31
2.3. Проєктування та реалізація алгоритмів системи	37
2.4. Створення функціоналу системи	42
Висновки до другого розділу	45
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ФУНКЦІОНАЛЬНИХ МОДУЛІВ СИСТЕМИ	47
3.1. Порядок встановлення та налаштування параметрів системи	47
3.2. Структура інтерфейсу та порядок взаємодії з платформою	52
3.3. Тестування роботи програмного продукту	68
Висновки до третього розділу	72
ВИСНОВКИ	74
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	75
ДОДАТКИ	78

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

CDN – Content Delivery Network

CI/CD – Continuous Integration / Continuous Deployment

EAN – European Article Number

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

JSONB – Binary JSON

JWT – JSON Web Token

MVC – Model-View-Controller

OEM – Original Equipment Manufacturer

ORM – Object-Relational Mapping

REST – Representational State Transfer

SKU – Stock Keeping Unit

SPA – Single Page Application

SQL – Structured Query Language

UI/UX – User Interface / User Experience

URL – Uniform Resource Locator

VIN – Vehicle Identification Number

БД – база даних

ПЗ – програмне забезпечення

СКБД – система керування базами даних

ТО – технічне обслуговування

ВСТУП

Актуальність теми зумовлена стрімким розширенням модельного ряду транспортних засобів та постійним ускладненням їхньої конструкції. У сучасному світі автомобіль потребує регулярного технічного обслуговування, що неухильно призводить до появи мільйонів унікальних артикулів запчастин. Традиційні методи підбору за паперовими каталогами або звичайним текстовим пошуком є вкрай неефективними через високу ймовірність помилок, замовлення несумісних деталей та невиправдану втрату часу. Хоча розвиток інформаційних технологій дозволив автоматизувати процеси ідентифікації деталей, більшість існуючих професійних рішень залишаються закритими корпоративними системами. Це створює гостру потребу в розробці доступних, гнучких та зручних інформаційних систем як для малого бізнесу, так і для кінцевих споживачів, які б могли ефективно обробляти великі масиви даних, миттєво знаходити аналоги та безпомилково перевіряти сумісність.

Метою дослідження є розробка алгоритмів та створення сучасної інформаційної системи підбору автомобільних запчастин, що забезпечує точну ідентифікацію деталей, надшвидкий повнотекстовий пошук та зручне керування каталогом продукції. Система має вирішувати завдання перевірки сумісності запчастин із багаторівневою ієрархією модифікацій транспортних засобів, а також надавати користувачам передові інструменти, такі як віртуальний гараж із функцією інтелектуального розкодування ідентифікаційних номерів автомобілів та автоматизований помічник регламентного технічного обслуговування.

Об'єктом дослідження виступають процеси автоматизації ідентифікації, пошуку та вибору комплектуючих для транспортних засобів у системах електронної комерції.

Предметом дослідження є використання сучасних інформаційних технологій, архітектурних шаблонів та алгоритмів пошуку за технічними параметрами для швидкої обробки великих каталогів товарів, побудови складних зв'язків між аналогами деталей та забезпечення надійної взаємодії клієнтської і серверної частин застосунку.

Для досягнення поставленої мети сформовано комплекс практичних завдань, що розпочинається з проведення аналізу вимог до систем ідентифікації автозапчастин та оцінки переваг і недоліків існуючих програмних рішень. На основі цього визначається архітектура системи за принципами суворого розділення бізнес-логіки та обґрунтовується вибір інструментальних засобів: сучасних платформ для розробки програмних інтерфейсів, надійних реляційних баз даних та спеціалізованих пошукових рушіїв. Наступні кроки включають розробку інформаційного забезпечення бази даних, проєктування алгоритмів фільтрації, розрахунку метрик якості аналогів та створення інтуїтивно зрозумілого інтерфейсу клієнтської частини. Завершальним етапом є інтеграція зовнішніх сервісів, розгортання архітектури та багаторівневе тестування застосунку для підтвердження його високої продуктивності, масштабованості та безпеки в умовах сучасного ринку.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД СУЧАСНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1. Постановка завдання

Розробка інформаційної системи у вигляді веб-застосунку для підбору автомобільних запчастин є важливим та актуальним завданням, що спрямоване на створення зручної, швидкої та ефективної онлайн-платформи для автовласників, працівників сервісних центрів та магазинів автозапчастин. Суть проєкту полягає у забезпеченні користувачів комплексним набором інструментів для точного пошуку деталей за артикулами та крос-номерами, перевірки їх сумісності з конкретними модифікаціями транспортних засобів, інтелектуального аналізу регламентного технічного обслуговування, а також у наданні розширених можливостей адміністрування для підтримки актуальності та повноти бази даних платформи. Ключовим завданням проєкту є побудова системи, яка не лише виконуватиме роль електронного каталогу запчастин, а виступатиме інтелектуальним помічником у процесі підбору комплектуючих. Сучасний автомобільний ринок характеризується надзвичайно великою кількістю технічних модифікацій транспортних засобів, і навіть незначна відмінність у комплектації двигуна чи трансмісії може визначати принципово різний набір сумісних запчастин. Тому критично важливим є забезпечення точної ідентифікації деталей через систему перехресних номерів та автоматизований пошук аналогів різних рівнів якості та цінових категорій.

Основним елементом майбутньої системи має стати модуль реєстрації та авторизації, який підтримуватиме безпечний доступ до персонального профілю із використанням механізму JWT-токенів [9]. Необхідно передбачити два незалежних шляхи авторизації: класичну реєстрацію з електронною поштою та паролем, а також швидку авторизацію через облікові записи сторонніх сервісів. Система повинна забезпечувати відновлення паролю через відправку тимчасового коду на електронну пошту, що вимагає інтеграції з SMTP-сервісом. Користувачі повинні мати можливість шукати автозапчастини за назвою, артикулом, оригінальним номером або довільними ключовими словами через

двоканальну систему пошуку. Перший канал передбачає пошук по реляційній базі даних із підтримкою автодоповнення та нечіткого зіставлення, тоді як другий канал вимагає імплементації повнотекстового пошуку через спеціалізований пошуковий движок, який забезпечить надшвидку відповідь на складні запити з підтримкою зваженого ранжування за релевантністю полів [6, 30]. Для забезпечення високої точності пошуку навіть при неповних запитах система має індексувати артикули, назви, бренди, категорії, крос-номери та технічні характеристики запчастин.

Важливою складовою системи необхідно зробити функціонал віртуального гаража, що дозволить авторизованому користувачу додавати до свого профілю конкретну модифікацію автомобіля, вказуючи VIN-код, поточний пробіг та персональне ім'я транспортного засобу. При цьому слід реалізувати автоматичне декодування VIN-коду через зовнішні сервіси, що дозволить ідентифікувати марку, модель, рік випуску, тип двигуна та інші параметри безпосередньо з коду. Система має виконувати інтелектуальне зіставлення отриманих даних з локальним довідником транспортних засобів у базі даних для автоматизації процесу та мінімізації помилок користувача. На основі збережених автомобілів у гаражі передбачається розробка інтелектуального помічника технічного обслуговування, який аналізуватиме поточний пробіг автомобіля та зіставлятиме його з ієрархічною системою регламентів. При наближенні терміну заміни система повинна автоматично формувати лист із переліком рекомендацій та надсилати його на електронну пошту користувача, додаючи прямі посилання на каталог відповідних запчастин.

На сайті необхідно спроектувати багатокритеріальний каталог із ієрархічним деревом категорій, підтримкою вкладених підкатегорій, фільтрацією за технічними характеристиками, брендами та рівнями якості, а також системою пагінації. Система має підтримувати автоматичну агрегацію доступних фільтрів на основі фактичних характеристик запчастин у обраній категорії. Окрему увагу слід приділити розробці модуля інтерактивних діаграм вузлів автомобіля з підтримкою ієрархії від групи вузлів до конкретної діаграми

з прив'язкою до модифікації транспортного засобу. Кожна діаграма повинна містити зображення схеми вузла та набір інтерактивних точок, які позиціонуватимуться за координатами і пов'язуватимуться з конкретною запчастиною, що дозволить візуально ідентифікувати потрібну деталь та швидко переходити до її картки. Для наповнення цих діаграм планується інтеграція з зовнішніми ресурсами. Також передбачається реалізація модуля партнерських локацій для пошуку автомагазинів та автосервісів через інтеграцію з картографічними API та сервісами геокодування, з підтримкою пошуку за координатами чи назвою міста та фільтрацією за типом.

Модуль відгуків та рейтингів повинен дозволяти авторизованим користувачам залишати оцінки, коментарі та вести діалоги через ієрархічну систему відповідей із автоматичним розрахунком середнього рейтингу. Система історії пошуку має зберігати та синхронізувати запити і переглянуті товари між локальним сховищем браузера та серверною базою даних. Модуль пошуку аналогів потребує реалізації алгоритмів для знаходження прямих та зворотних зв'язків взаємозамінності, а також розробки спеціального алгоритму оцінки найкращого співвідношення ціни та якості, який враховуватиме рівень якості бренду, позначку оригінальності та ціну товару. Для повноцінного управління всіма аспектами системи необхідно створити адміністративну панель зі спеціалізованим інтерфейсом та рольовою моделлю доступу. Вона повинна включати модулі управління користувачами, каталогом запчастин із функцією автозаповнення через зовнішні API, ієрархічними категоріями, довідниками транспортних засобів, інтерактивними діаграмами та аналітичною панеллю на основі даних історії пошуку.

Весь цей функціонал має бути надійно захищений від помилок введення та спроб зламу, гарантуючи швидкий доступ до великих обсягів інформації. Інтерфейс сайту потрібно зробити інтуїтивно зрозумілим, сучасним та повністю адаптивним для різних екранів. Технічна реалізація вимагає вибору ефективного стеку технологій, проектування розгалуженої реляційної структури бази даних, впровадження надійних засобів безпеки, таких як хешування паролів, захист від

ін'єкцій та міжсайтового скриптингу. Невід'ємною частиною проєкту має стати налаштування безперервної інтеграції та доставки, автоматичного лінтингу, перевірки безпеки залежностей та комплексного тестування. Завершальним етапом стане налаштування середовища для розгортання застосунку з підтримкою контейнеризації для інфраструктурних залежностей, що гарантуватиме стабільну, швидку й безпечну роботу сучасної веб-платформи для підбору автозапчастин. Результатом реалізації проєкту стане сучасна платформа, що перевершуватиме аналоги завдяки унікальній комбінації інтелектуальних функцій, продуманій архітектурі та захищеним програмним інтерфейсам.

1.2. Аналіз існуючого програмного забезпечення

Перш ніж розпочати розробку інформаційної системи, важливо провести ґрунтовний аналіз існуючих рішень на ринку AutoTech. Це дозволяє оцінити доступні функціональні можливості, визначити сильні та слабкі сторони конкурентів, оцінити рівень технологічної реалізації, а також знайти ідеї для покращення власного сервісу. Аналіз аналогів допомагає зрозуміти поточний стан галузі та сформуванати конкурентоспроможну стратегію розвитку продукту, що відповідає очікуванням цільової аудиторії.

Під час аналізу ринку сервісів для ідентифікації та підбору автозапчастин було виокремлено три провідні платформи: Exist.ua, Autodoc.ua та TecDoc. Нижче наведено детальний розгляд кожного з них з акцентом на їхні переваги та недоліки.

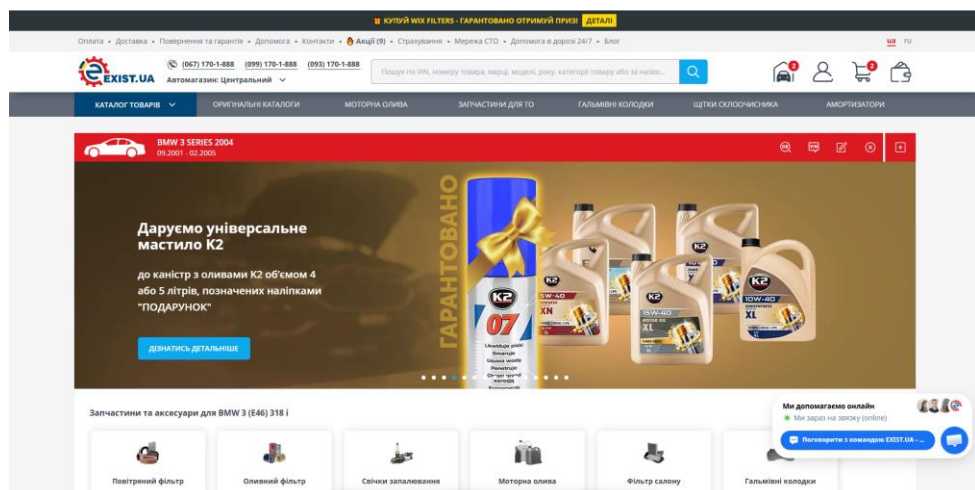


Рис. 1.1 – Вигляд головної сторінки веб-сайту Exist.ua

Серед вагомих переваг системи — величезна база даних крос-номерів, наявність деталізованих оригінальних схем вузлів автомобіля, гнучка система фільтрації та можливість професійного підбору запчастин персональним менеджером. Також варто зазначити потужну систему VIN-декодування, яка дозволяє за ідентифікаційним номером автомобіля точно визначити конфігурацію транспортного засобу та отримати перелік оригінальних запчастин. Платформа пропонує інтегровану логістику з мережею складів по всій Україні, що забезпечує швидку доставку.

AUTODOC Доставка Покупки Про магазин Контакты

Каталог В корзину

Детали кузова
Автозапчасти
Электрика та освітлення
Масла, тех. рідини та автохімія
Шини та диски
Зовнішні/внутрішні елементи авто
Інструменти та обладнання
Турки і відпочинок
Сільськогосподарське обладнання

Отримай статус клубного клієнта до кінця 2026 року

КЛУБНИЙ КЛІЄНТ

Зроби замовлень на **1500 грн**
у період з 16.03.2026 по 30.04.2026

Autodoc.ua — це сучасний сервіс підбору та продажу запчастин, який робить акцент на швидку доставку та інтеграцію з власною мережею станцій технічного обслуговування. Головними перевагами платформи є інтуїтивно зрозумілий, сучасний та лаконічний інтерфейс, який дозволяє швидко знаходити потрібні товари. Також варто відзначити висококваліфікований автоматизований підбір запчастин фахівцями та ефективну систему логістики з відстеженням статусів замовлень. Autodoc пропонує зручну систему фільтрації за маркою та моделлю автомобіля, а також інтеграцію з відеоінструкціями з ремонту, що допомагає користувачам самостійно виконувати нескладні роботи з обслуговування.

До недоліків сервісу можна віднести цінову політику, яка є вищою за середньоринкову через включення у вартість супутніх сервісних послуг. Крім того, порівняно з Exist.ua, платформа пропонує меншу кількість доступних аналогів, особливо для рідкісних або застарілих моделей автомобілів. Відсутня функція інтерактивних діаграм вузлів та схем автомобіля, що ускладнює візуальну ідентифікацію потрібної деталі для непідготовленого користувача. Система не надає функціоналу «Віртуального гаража» з відстеженням регламентів ТО та проактивними рекомендаціями на основі пробігу.

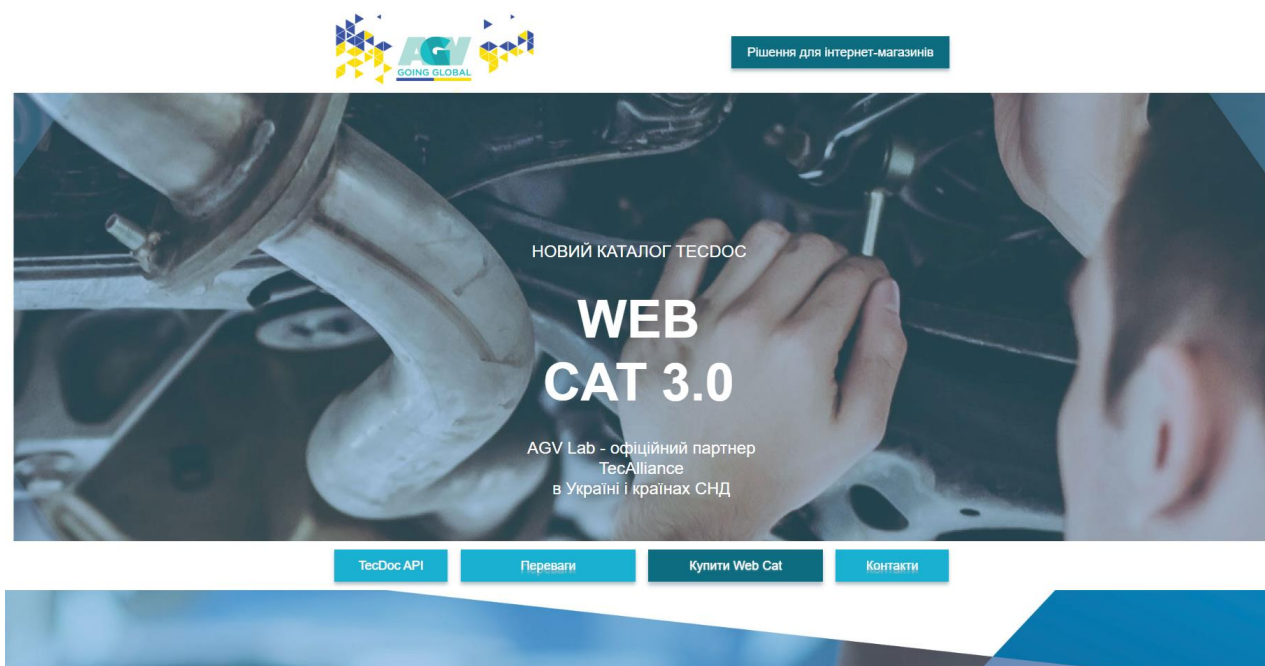


Рис. 1.3 – Вигляд головної сторінки платформи TecDoc

TecDoc від компанії TecAlliance не є класичним інтернет-магазином; це глобальний галузевий стандарт і потужна інформаційна база даних, яку використовують розробники, виробники та продавці по всьому світу. Беззаперечною перевагою платформи є найвища на ринку точність ідентифікації деталей за VIN-кодом та технічними характеристиками транспортного засобу. Система має офіційну підтримку від виробників запчастин та надає можливості глибокої інтеграції через веб-API. TecDoc забезпечує стандартизовану класифікацію деталей та їх характеристик, що робить його незамінним інструментом для професійних учасників ринку.

Водночас суттєвими недоліками для локальних ринків та невеликих проєктів є висока вартість передплати — орієнтація відбувається виключно на B2B-сегмент та надзвичайна складність інтеграції через масивність документації. Також система не має вбудованої інтеграції з локальними складами на території України, будучи лише інформаційним довідником. Відсутній функціонал для кінцевого споживача: немає системи відгуків, рейтингів, персоналізованих рекомендацій чи інтерактивних схем для візуальної навігації. Інтерфейс TecDoc є суто технічним та не адаптованим для використання непрофесійними користувачами.

Таблиця 1.1

Порівняння аналогів систем підбору автозапчастин

Критерій / Сервіс	Exist.ua	Autodoc.ua	TecDoc	AutoParts System
Пошук (Elasticsearch)	Ні	Ні	Частково	Так
VIN-декодування	Так	Частково	Так	Так
Віртуальний гараж	Ні	Ні	Ні	Так
Помічник ТО	Ні	Ні	Ні	Так
Інтерактивні діаграми	Так	Ні	Ні	Так
Система аналогів	Так	Частково	Так	Так
Карта магазинів (OSM)	Ні	Ні	Ні	Так
Google OAuth	Ні	Ні	Ні	Так
Система відгуків	Обмежена	Так	Ні	Так
Відкритий API	Ні	Ні	Так (платний)	REST API

Проведений аналіз свідчить про наявність чіткого ринкового запиту на програмне рішення, яке б поєднувало високий рівень точності та структурованості даних, притаманні професійним каталогам на кшталт TecDoc, із зручністю інтерфейсу, швидкістю пошуку та орієнтацією на кінцевого користувача, характерними для Autodoc [8]. Жоден з існуючих аналогів не задовольняє всі ці потреби одночасно без суттєвих недоліків для звичайного користувача чи локального бізнесу. Зокрема, жодна з проаналізованих платформ не пропонує комплексного поєднання таких функцій, як інтелектуальний помічник ТО з імейл-сповіщеннями, інтерактивні діаграми вузлів з прив'язкою до запчастин, пошук автомагазинів через OpenStreetMap та алгоритм оцінки найкращого співвідношення якості та ціни (ValueScore).

Таким чином, існує об'єктивна потреба у розробці комплексної інформаційної системи, що об'єднає точність професійних галузевих баз даних із практичністю сучасних веб-застосунків. Враховуючи проведений аналіз, при створенні власної системи доцільно орієнтуватися на простоту взаємодії Autodoc та глибину даних Exist, розширюючи їх унікальним функціоналом «Віртуального гаража» з помічником ТО, інтерактивними діаграмами вузлів, інтеграцією з OpenStreetMap для пошуку партнерів та відкритими можливостями інтеграції через REST API. Це підтверджує актуальність розробки запропонованого програмного продукту як конкурентоспроможного та зручного рішення, що виходить за межі традиційного онлайн-каталогу запчастин.

1.3. Вибір архітектури системи

Вибір архітектури веб-орієнтованої системи визначає її продуктивність, масштабованість, безпеку та простоту подальшої модифікації. Правильна архітектура мінімізує технічні ризики та оптимізує ресурси, що є критично важливим для великих каталогів автозапчастин зі складними зв'язками сумісності, множинними інтеграціями з зовнішніми API та вимогами до надшвидкого пошуку.

У проєкті реалізовано класичну клієнт-серверну архітектуру на базі RESTful API [12]. Цей підхід стандартизує обмін даними у форматі JSON між клієнтською (Frontend) та серверною (Backend) частинами, що забезпечує гнучкість системи. Це спрощує інтеграцію нових функцій і дозволяє незалежно масштабувати або оновлювати клієнтський інтерфейс та серверну логіку (рис. 1.4). Розділення відповідальностей є фундаментальним принципом цієї архітектури, що забезпечує повну автономність обох частин додатка, дозволяючи їм взаємодіяти виключно через захищені програмні інтерфейси та оптимізувати навантаження на серверні потужності й бази даних незалежно одна від одної.

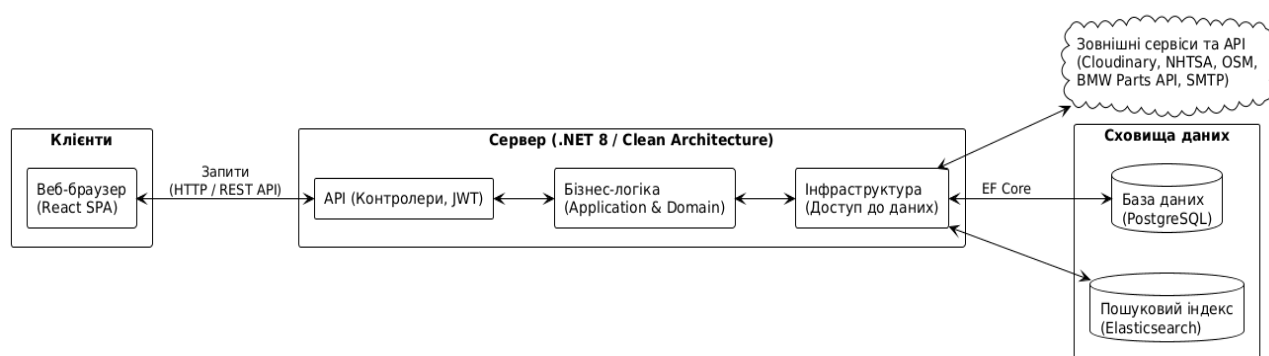


Рис. 1.4 – Клієнт-серверна архітектура в поєднанні з модульним монолітом

Серверну частину побудовано за архітектурним шаблоном Clean Architecture із чітким розділенням на рівні домену, застосунку, інфраструктури та API [4]. Платформою розробки обрано .NET 8 з використанням мови C# 12 та фреймворку ASP.NET Core [1,2,3]. Цей підхід забезпечує високу продуктивність, строгу типізацію та ефективне використання потужної вбудованої системи впровадження залежностей.

Використання обраного фреймворку дозволяє структурувати код через контролери, ефективно обробляти мережеві запити та гнучко налаштовувати механізми авторизації через JWT-токени і політики безпеки CORS. Інфраструктурний шар повністю інкапсулює зовнішні інтеграції. До них належить робота з базами даних і пошуковими системами, взаємодія з картографічними сервісами, звернення до зовнішніх API для декодування VIN-

кодів та каталогів автозапчастин, а також парсинг даних, хмарне зберігання зображень і відправка електронних листів.

Взаємодія з реляційною базою даних PostgreSQL реалізована через Entity Framework Core з провайдером Npgsql [5,15]. Це гарантує захист від вразливостей баз даних завдяки параметризованим запитам, спрощує побудову логіки перевірки сумісності деталей та підвищує загальну надійність. Управління схемою даних здійснюється через механізм автоматичних міграцій, що забезпечує безпечне версіонування структури у робочому середовищі.

Для забезпечення високої швидкості повнотекстового пошуку архітектуру доповнено пошуковим рушієм Elasticsearch. Його індекс містить денормалізовану інформацію про ключові характеристики запчастин і оновлюється синхронно при будь-яких змінах у каталозі. Для стабільності та зручності розгортання цей сервіс ізольовано працює у вигляді контейнера за допомогою Docker.

Клієнтську частину реалізовано як односторінковий застосунок (SPA) за допомогою бібліотеки React та збірника Vite [13, 20]. Вона взаємодіє із сервером через REST API, використовуючи HTTP-клієнт Axios із налаштованими перехоплювачами для автоматичної передачі JWT-токенів. Для маршрутизації застосовується React Router v6 із підтримкою вкладених шляхів для адміністративної панелі.

Архітектура фронтенду базується на адаптованому патерні MVVM. Рівень моделі відповідає за обмін даними із сервером. Логіку управління станом, обробку асинхронних операцій та трансформацію даних інкапсульовано в кастомних React-хуках, що виконують роль ViewModel [7,16]. Візуалізація реалізована через перевикористовувані компоненти, які отримують готові дані та відповідають виключно за рендеринг. Глобальний стан авторизації керується через React Context API. Такий розподіл обов'язків чітко відокремлює бізнес-логіку від інтерфейсу, що значно підвищує модульність і тестованість коду (рис. 1.5).

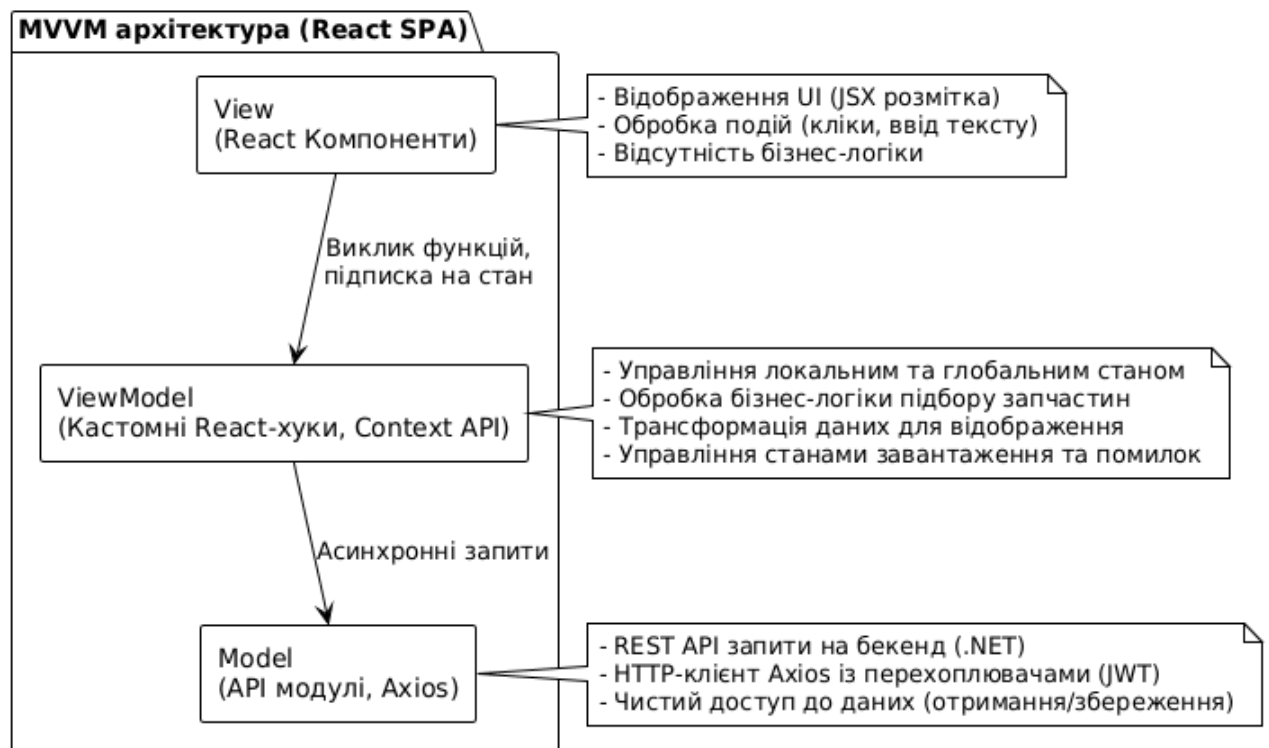


Рис. 1.5 – Візуалізація патерну взаємодії компонентів у React-застосунку

Для оптимізації роботи та зниження навантаження на сервер систему розширено інтеграціями із хмарними сервісами та зовнішніми програмними інтерфейсами. Зокрема, платформа Cloudinary забезпечує зберігання та миттєву доставку оптимізованих зображень запчастин, логотипів і діаграм через глобальну мережу доставки контенту без фізичного завантаження файлів на бекенд [11]. Інтеграція з сервісом NHTSA дозволяє автоматично декодувати VIN-коди транспортних засобів, а зовнішній інтерфейс BMW Parts Catalog використовується для наповнення каталогу актуальними технічними даними та інформацією про сумісність. Геопошук магазинів і станцій технічного обслуговування реалізовано завдяки сервісам Overpass та Nominatim [22], тоді як спеціальний парсер BMWFans відповідає за автоматичне завантаження інтерактивних схем вузлів. Завдяки такій архітектурі система забезпечує чітке розділення відповідальностей, ефективне управління складністю проекту, можливість незалежного масштабування компонентів та стабільну продуктивність під високим навантаженням.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вибір інструментальних засобів для розробки сучасної інформаційної системи є одним із найвідповідальніших етапів проєктування, оскільки він безпосередньо визначає межі масштабованості, рівень безпеки, продуктивність під навантаженням та економічну доцільність подальшого супроводження платформи. Враховуючи специфіку електронної комерції у сфері автозапчастин — наявність мільйонів товарних позицій, складні ієрархічні зв'язки сумісності та необхідність інтеграції з багатьма зовнішніми сервісами — технологічний стек мав формуватися на базі перевірених, продуктивних та безпечних рішень [4, с. 48].

На етапі вибору платформи для розробки бекенду розглядалися три популярні сучасні рішення: Node.js (Express), Java (Spring Boot) та .NET 8 (ASP.NET Core). Платформа Node.js, незважаючи на високу швидкість розробки та неблокуючу архітектуру, має недоліки при обробці складних обчислювальних операцій та строгій типізації великих доменних моделей. Java Spring Boot є потужним ентєрпрайз-рішенням, проте вимагає значних ресурсів оперативної пам'яті та характеризується довшим часом «холодного старту».

У результаті для розробки серверної частини було обрано платформу .NET 8.0 (реліз із довгостроковою підтримкою LTS) та об'єктно-орієнтовану мову програмування C# 12 [1, с. 15]. Цей вибір забезпечує низку критичних переваг, серед яких ключове місце посідає висока продуктивність, оскільки вбудований вебсервер Kestrel у .NET 8 демонструє одні з найкращих показників у незалежних бенчмарках (наприклад, TechEmpower) за кількістю оброблюваних HTTP-запитів на секунду. Окрім цього, суворі статична типізація дозволяє виявляти більшість помилок архітектури та несумісності типів ще на етапі компіляції, що є критично важливим під час реалізації складної бізнес-логіки перевірки сумісності запчастин із транспортними засобами [3, с. 112]. Вагомим фактором також є розвинена екосистема та архітектура платформи, де вбудована система впровадження залежностей (Dependency Injection) «з коробки»

стимулює дотримання принципів SOLID та Clean Architecture, тим самим забезпечуючи слабку зв'язність між контролерами, сервісами та репозиторіями [18, с. 81].

Для побудови інтерфейсу користувача (UI) розглядалися фреймворки Angular, Vue.js та бібліотека React. Вибір було зроблено на користь React 18 [24]. На відміну від громіздкого Angular, React пропонує гнучкий декларативний підхід. Використання механізму Virtual DOM дозволяє мінімізувати кількість прямих звернень до реального DOM-дерева браузера. Під час оновлення стану (наприклад, при застосуванні динамічних фільтрів у каталозі автозапчастин) React обчислює різницю між поточним та новим станом віртуального дерева (алгоритм Reconciliation) і точково оновлює лише змінені елементи, гарантуючи плавність інтерфейсу [16, с. 55].

В якості інструмента збірки фронтенду замість традиційного Webpack було впроваджено Vite [25]. Vite використовує esbuild, написаний на мові Go, що пришвидшує процес локальної збірки у 10–100 разів порівняно з інструментами на базі JavaScript. Завдяки механізму Hot Module Replacement (HMR) розробка клієнтської частини стає максимально інтерактивною. Візуальна стилізація реалізована через підхід Utility-First за допомогою фреймворку TailwindCSS, що дозволило відмовитися від написання тисяч рядків кастомного CSS-коду на користь перевикористовуваних атомарних класів безпосередньо у JSX-розмітці.

Робота з каталогами запчастин вимагає збереження як жорстко структурованих даних (зв'язки автомобілів), так і гнучких неструктурованих атрибутів (специфічні технічні параметри деталей). Для обґрунтування вибору було проведено порівняльний аналіз популярних СКБД (табл. 1.2).

Таблиця 1.2

Порівняльний аналіз систем керування базами даних

Критерій	MySQL 8.0	MongoDB	PostgreSQL 15
Модель даних	Реляційна	Документно-орієнтована	Об'єктно-реляційна
Транзакційність (ACID)	Повна підтримка	Часткова (на рівні документа)	Повна підтримка

Робота зі специфікаціями	Базова підтримка JSON	Нативна підтримка (BSON)	Високооптимізований формат JSONB з індексацією
Підтримка масивів	Немає нативної підтримки	Нативна підтримка	Нативна підтримка

На основі аналізу основним сховищем обрано PostgreSQL 15 [5, с. 59]. Вирішальним фактором стала наявність бінарного формату JSONB, що дозволяє зберігати гнучкі характеристики товарів (різьба, напруга, кількість контактів) та індексувати їх за допомогою GIN-індексів для швидкого пошуку, не порушуючи при цьому реляційну цілісність бази даних. Також PostgreSQL нативно підтримує масиви, що ідеально підходить для збереження OEM-номерів та EAN-кодів [14, с. 110]. Взаємодія серверної частини з PostgreSQL здійснюється через ORM-систему Entity Framework Core, що нівелює ризики SQL-ін'єкцій та забезпечує зручне управління міграціями бази даних [23].

Звичайні SQL-запити з оператором LIKE є неефективними при пошуку серед мільйонів артикулів, особливо якщо користувач допускає друкарські помилки або використовує спецсимволи. Тому архітектуру було доповнено пошуковим рушієм Elasticsearch 8.12 [30]. Він базується на концепції інвертованого індексу та реалізує алгоритми зваженого ранжування (BM25) і нечіткого зіставлення (Fuzzy matching), що дозволяє знаходити потрібну запчастину за мілісекунди навіть при неповному збігу [17, с. 45].

Зберігання медіафайлів (фотографій запчастин) на власному сервері є економічно та технічно неефективним, тому цю функцію делеговано спеціалізованому хмарному сервісу Cloudinary. Він автоматично конвертує зображення у сучасний формат WebP та доставляє їх клієнтам через мережу CDN (Content Delivery Network), суттєво зменшуючи час завантаження сторінок каталогу [11, с. 93].

Безпековий контур реалізовано через протокол безстанової авторизації за допомогою JSON Web Tokens (JWT) з криптографічним підписом HMAC-SHA256. Паролі користувачів захищені алгоритмом хешування BCrypt із застосуванням унікальної «солі» (salt) для кожного запису, що унеможливорює злам методом перебору за словником (rainbow tables) [9, с. 91]. Додатково інтегровано протокол OAuth 2.0 для забезпечення безпечного та швидкого входу через облікові записи Google.

Організація процесу розробки спирається на методологію DevOps. Контейнеризація за допомогою Docker Compose гарантує, що база даних та пошуковий рушій працюватимуть ідентично як на локальному комп'ютері розробника, так і на продуктовому сервері [19, с. 115]. Автоматизація процесів збірки та тестування реалізована через GitHub Actions, де налаштовано конвеєр CI/CD. До нього інтегровано платформу SonarCloud для статичного аналізу коду, що дозволяє автоматично виявляти технічний борг та вразливості на етапі створення запиту на злиття коду (Pull Request) [10, с. 37].

Обраний комплекс інструментальних засобів, екосистема .NET, потужність бази даних PostgreSQL у поєднанні з Elasticsearch та хмарними сервісами повністю покривають специфічні вимоги електронної комерції у сфері автозапчастин. Такий підхід гарантує створення безпечної, швидкої та легко масштабованої системи, здатної обробляти значні масиви технічної інформації.

Висновки до першого розділу

У першому розділі визначено основні завдання та технічні вимоги для розробки комплексної інформаційної системи підбору автозапчастин. Проведено аналіз існуючих рішень на ринку AutoTech (Exist.ua, Autodoc.ua, TecDoc) із визначенням їх функціональності, переваг і недоліків, що підтвердило необхідність створення платформи з унікальним поєднанням інтелектуального помічника ТО, інтерактивних діаграм та розширеного пошуку.

Архітектура системи обрана з урахуванням специфіки роботи з великими обсягами технічних даних та зовнішніми інтеграціями: серверна частина реалізована за принципами Clean Architecture на базі .NET 8 (ASP.NET Core Web

API), що забезпечує високу продуктивність, строгу типізацію та незалежність бізнес-логіки; клієнтська частина побудована за клієнт-серверною моделлю (SPA) з використанням бібліотеки React та збірника Vite. У клієнтській частині застосовується архітектурний підхід MVVM через використання React-хуків та Context API для ефективного розділення логіки підбору, стану даних та візуального інтерфейсу, що полегшує підтримку і подальший розвиток проєкту.

Для практичної реалізації обрано оптимальний стек технологій: PostgreSQL 15 як надійну реляційну СКБД для управління складними зв'язками між запчастинами та автомобілями, Entity Framework Core як ефективний інструмент взаємодії з даними, Elasticsearch — для надшвидкого повнотекстового пошуку, React — для побудови динамічного інтерфейсу, та Cloudinary — для оптимізованого хмарного зберігання зображень. Базовий комплекс доповнено інтеграціями з зовнішніми API та контейнеризацією Docker. Даний інструментарій повністю відповідає поставленим завданням та сучасним стандартам веброзробки.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ

2.1. Аналіз вимог та специфікація системи

Аналіз вимог охоплює визначення та уточнення як функціональних, так і нефункціональних характеристик програмного забезпечення. Він включає оцінку часу реакції системи на дії користувачів, опис можливих сценаріїв її використання, а також створення загальної схеми елементів системи з поясненням їх структури, взаємодії та методів функціонування. Такий підхід дозволяє досягти чіткого розуміння потреб автовласників та адміністраторів, а також визначити технічні параметри, необхідні для розробки зручного та ефективного каталогу підбору автозапчастин з інтелектуальними можливостями.

У межах інформаційної системи передбачено чітке розмежування прав доступу, що базується на трьох основних рівнях: неідентифікований відвідувач, авторизований користувач та адміністратор. Неідентифікований користувач (гість) має базовий набір функцій, який дозволяє здійснювати пошук запчастин за артикулом, назвою або оригінальним номером, переглядати каталог із фільтрацією за категоріями та брендами, а також ознайомлюватися з детальною інформацією про товари, включаючи характеристики, фотографії, аналоги та відгуки. Додатково гостю доступний перегляд каталогу транспортних засобів та інтерактивних діаграм вузлів. Для отримання розширених можливостей необхідно пройти процедуру класичної реєстрації за допомогою електронної пошти або швидкої авторизації через сервіси Google.

Після успішної авторизації користувач платформи отримує доступ до повноцінного клієнтського функціоналу, ключовим елементом якого є модуль віртуального гаража. Цей інструмент дозволяє зберігати конфігурації власних автомобілів шляхом ручного вибору параметрів або через автоматичне декодування VIN-коду із залученням зовнішніх програмних інтерфейсів. На основі вказаного пробігу збереженого транспортного засобу система автоматично аналізує регламенти технічного обслуговування та завчасно надсилає персоналізовані імейл-рекомендації щодо планової заміни компонентів. Авторизований клієнт може здійснювати швидкий повнотекстовий

пошук через Elasticsearch, автоматично перевіряти сумісність знайдених деталей зі своїми автомобілями, аналізувати аналоги за алгоритмом оцінки співвідношення ціни та якості, знаходити найближчі автомагазини чи станції технічного обслуговування, брати участь в обговореннях через ієрархічну систему відгуків, а також керувати власним профілем та історією пошукових запитів.

Найвищий рівень доступу в системі надається адміністратору, який здійснює повне управління платформою, контролює її наповнення та аналізує статистичні показники. Розподіл функціональних можливостей адміністратора наведено у таблиці нижче.

Таблиця 2.1

Функціональні можливості адміністратора системи

Модуль системи	Опис можливостей управління
Користувачі та аналітика	Призначення системних ролей користувачам; перегляд аналітичного дашборду зі статистикою популярних товарів, категорій та автомобілів.
Каталог запчастин	Створення, редагування та видалення товарів; автоматичне заповнення характеристик, зображень і OEM-номерів через зовнішні API; синхронізація аналогів; управління зв'язками сумісності деталей.
Довідники автомобілів	Управління ієрархією транспортних засобів: марками, моделями, модифікаціями з чіткою прив'язкою до типів двигунів, трансмісій, приводів та кузовів.
Інтерактивні діаграми	Створення ієрархії груп вузлів; завантаження SVG-схем; розстановка інтерактивних точок (hotspots) із прив'язкою до OEM-номерів; імпорт готових діаграм із зовнішніх парсерів.
Технічне обслуговування	Управління регламентами ТО з підтримкою каскадної специфічності (від загальних правил до конкретної модифікації); налаштування категорій обслуговування.
Партнерські локації	Ручне створення точок на карті; автоматизований імпорт автомагазинів та сервісів з OpenStreetMap; налаштування розумної фільтрації за країнами-виробниками авто.
Класифікатори	Управління деревом категорій каталогу; адміністрування брендів із зазначенням країни походження, рівня якості та позначки оригінальності.

Нефункціональні вимоги описують ключові властивості та показники якості системи, що безпосередньо впливають на її ефективність, надійність і зручність використання. Одним із пріоритетних аспектів є якість

користувачького досвіду. Інтерфейс розроблено таким чином, щоб забезпечити інтуїтивну зрозумілість процесу підбору деталей навіть для автовласників без спеціальних технічних знань. Візуальне оформлення виконано у сучасному стилі, при цьому гарантується повна адаптивність для мобільних пристроїв із підтримкою навігаційних елементів типу бокового меню та hamburger-кнопки.

Продуктивність і масштабованість є критичними характеристиками, враховуючи необхідність обробки сотень тисяч записів у базах даних каталогу. Архітектура гарантує високу швидкість відгуку завдяки використанню пошукового движка Elasticsearch із алгоритмами зваженого ранжування та нечіткого зіставлення, що нівелює проблему великих обсягів індексу. Вимоги до безпеки реалізуються через обов'язкове використання захищеного протоколу HTTPS. Авторизація базується на механізмі JWT-токенів із криптографічним підписом та терміном дії 24 години, системі хешування паролів BCrypt і підтримці протоколу Google OAuth 2.0, а для відновлення доступу застосовується тимчасовий шестизначний код [9]. Безпека на рівні застосунку забезпечується суворим розмежуванням прав доступу на основі ролей (Admin, Customer), налаштуванням CORS-політик для захисту від міжсайтової підробки запитів (CSRF) та параметризацією запитів до бази даних для унеможливлення SQL-ін'єкцій.

Система має безпомилково функціонувати в усіх основних сучасних веб-браузерах та на екранах різних форматів. Безперебійність роботи платформи контролюється через регулярний автоматичний моніторинг ендпоінтів, при цьому час на планове технічне обслуговування не повинен перевищувати 3% від загального часу роботи на місяць.

Часові показники реакції системи на дії користувача виступають найважливішим критерієм її продуктивності. Для забезпечення комфортної взаємодії з платформою встановлено нормативні обмеження швидкодії, які наведено у таблиці.

Вимоги до часових характеристик реакції системи

Категорія операцій	Опис дії або процесу	Максимальний час виконання
Типові дії користувача	Прямий пошук деталі за артикулом (Elasticsearch)	2 с
	Автодоповнення пошукового запиту (suggest)	1 с
	Перехід між сторінками каталогу	3 с
	Завантаження оптимізованого зображення через CDN	3 с
Аутентифікація	Процес реєстрації або авторизації (включно з Google OAuth)	3 с
Складні дії користувача	Перевірка сумісності деталі з автомобілем (реляційний запит)	5 с
Складні дії користувача	Пошук аналогів із розрахунком метрики ValueScore	5 с
	Декодування VIN-коду через NHTSA API	8 с
	Пошук партнерських локацій через Overpass API	15 с
Адміністративні функції	Автозаповнення даних запчастини через BMW Parts Catalog API	10 с
	Синхронізація бази аналогів запчастин	30 с
	Імпорт партнерських локацій з OpenStreetMap	30 с
	Повний процес ручного створення нової картки товару	3 хв

Специфікація варіантів використання розкриває ключові сценарії взаємодії різних категорій користувачів із платформою. Відображення архітектури цих процесів наведено на діаграмі прецедентів, яка деталізує загальну структуру доступних системних функцій [21].

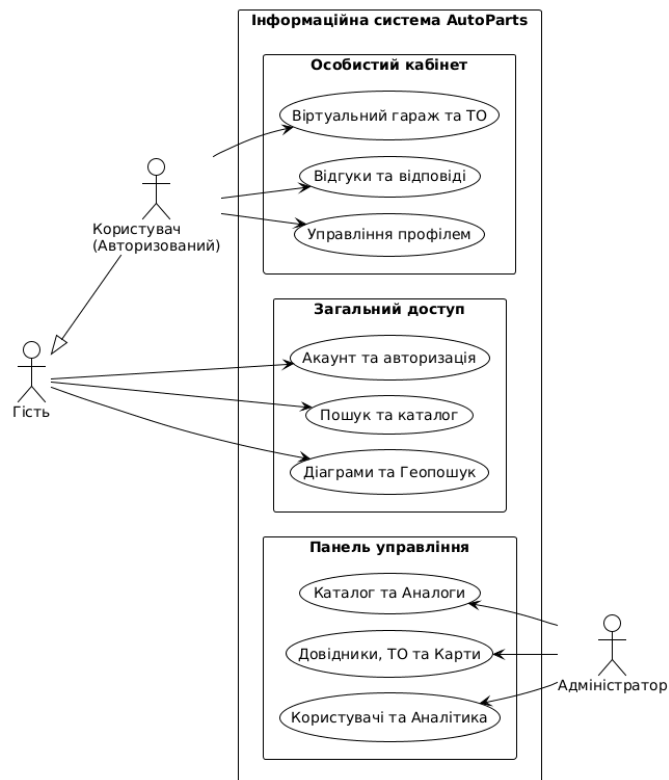


Рис. 2.1 – Варіанти використання системи

Архітектура серверної частини на базі .NET 8 Web API побудована за принципами Clean Architecture з модульним розділенням, що забезпечує високу гнучкість, масштабованість та легкість у підтримці. Кожен функціональний домен системи логічно винесено в окремий компонент, який містить контролери, сервіси, об'єкти передачі даних та доменні сутності. Це дозволяє інкапсулювати логіку кожного домену, використовуючи механізми впровадження залежностей для безпечної та ефективної взаємодії між модулями.

Таблиця 2.3

Модулі серверної частини (Clean Architecture)

Сутність	Опис
UsersController	Управління користувачами: Реєстрація, авторизація (Email, Google OAuth), хешування паролів (BCrypt), генерація JWT-токенів, відновлення доступу, розмежування ролей.
PartsController	Управління запчастинами: Повний цикл CRUD-операцій товарів, робота з характеристиками (JSONB), збереження OEM/EAN-номерів, ініціація індексації, обробка відгуків.
ElasticPartSearchService	Повнотекстовий пошук: Інтеграція з Elasticsearch: MultiMatch-запити, зважене ранжування, нечітке зіставлення (Fuzzy matching), автодоповнення запитів [17].

CatalogController	Каталог запчастин: Ієрархічна навігація каталогом, динамічна контекстна фільтрація, агрегація характеристик, пагінація результатів.
CategoriesController	Ієрархія категорій: Управління деревоподібною структурою каталогу за допомогою рекурсивних зв'язків.
BrandsController	Довідник брендів: Адміністрування виробників із підтримкою рівнів якості та позначок оригінальності деталей.
AdminVehiclesController, ModificationsController	Довідник автомобілів: Управління повною ієрархією транспортних засобів (марки, моделі, модифікації) з деталізацією параметрів двигунів та кузовів.
CompatibilityController	Перевірка сумісності: Управління реляційними зв'язками між запчастинами та модифікаціями автомобілів для точного підбору.
GarageController	«Віртуальний гараж»: Збереження автомобілів користувача, відстеження поточного пробігу.
NhtsaVinDecoderService	VIN-декодування: Інтеграція з NHTSA API, розкодування 17-символьних VIN-кодів, інтелектуальне зіставлення даних з локальною базою.
MaintenanceAssistantService	Помічник ТО: Аналіз пробігу з гаража, порівняння з регламентами, генерація та надсилання персоналізованих email-рекомендацій.
MaintenanceRulesController	Регламенти ТО: Управління правилами технічного обслуговування з підтримкою каскадної специфічності (від загальних до конкретної модифікації).
AnalogsController	Крос-номери (Аналоги): Двонаправлений пошук взаємозамінних деталей, розрахунок метрики співвідношення ціни та якості (ValueScore).
AdminPartSyncService	Синхронізація та автозаповнення: Автоматичний пошук аналогів через зовнішні API та заповнення карток товарів характеристиками, сумісностями й фото.
UploadController, CloudinaryImageStorage-Service	Хмарні медіадані: безпечне завантаження фотографій, генерація оптимізованих CDN-посилань через платформу Cloudinary.
DiagramsController, AdminDiagramsController	Інтерактивні діаграми: Управління ієрархією вузлів автомобіля, завантаження SVG-схем, створення інтерактивних точок (hotspots) із прив'язкою до OEM.
PartnerLocationsController	Партнерські локації: Геопошук магазинів і СТО через OpenStreetMap, імпорт локацій, фільтрація точок на карті.
SearchHistoryController	Історія пошуку: Збереження запитів користувачів, синхронізація з браузером, підготовка аналітики для дашборду.

2.2. Проєктування бази даних

Як основне сховище даних інформаційної системи підбору автозапчастин обрано об'єктно-реляційну систему керування базами даних PostgreSQL 15. Цей вибір обґрунтований її високою надійністю, продуктивністю та повноцінною підтримкою ACID-транзакцій, що є критично важливим для обробки великих масивів технічних каталогів та складної логіки перевірки сумісності [14]. Важливою архітектурною перевагою цієї бази даних є нативна підтримка масивів для збереження OEM-номерів та EAN-кодів, а також типу даних JSONB для гнучкої та ефективної роботи зі специфікаціями деталей. За своїми можливостями PostgreSQL не поступається комерційним аналогам і забезпечує бездоганну інтеграцію з ORM-фреймворком Entity Framework Core через провайдер Npgsql, що ідеально доповнює обраний технологічний стек .NET 8 [15].

Для забезпечення комплексного функціонування платформи було спроектовано розгалужену структуру, що налічує двадцять п'ять реляційних таблиць. Усі сутності логічно згруповані за доменами системи (рис. 2.2): управління користувачами та безпекою доступу (User, Role, PasswordResetToken), ведення віртуального гаража (UserVehicle), адміністрування товарного каталогу та сумісностей (Brand, Category, Part, PartAnalog, PartCompatibility, Review). Окремий глибоко структурований блок відповідає за збереження повної ієрархії транспортних засобів (Make, Model, Modification, Engine, Transmission, BodyType, DriveType). Додатково реалізовано таблиці для управління регламентами технічного обслуговування (MaintenanceRule, MaintenanceCategory), підтримки ієрархії інтерактивних діаграм вузлів (VehicleGroup, VehicleSubGroup, DiagramImage, DiagramHotspot), адміністрування геоданих партнерських локацій (PartnerLocation, PartnerLocationCountry) та збереження аналітичної історії пошуку (SearchHistory).

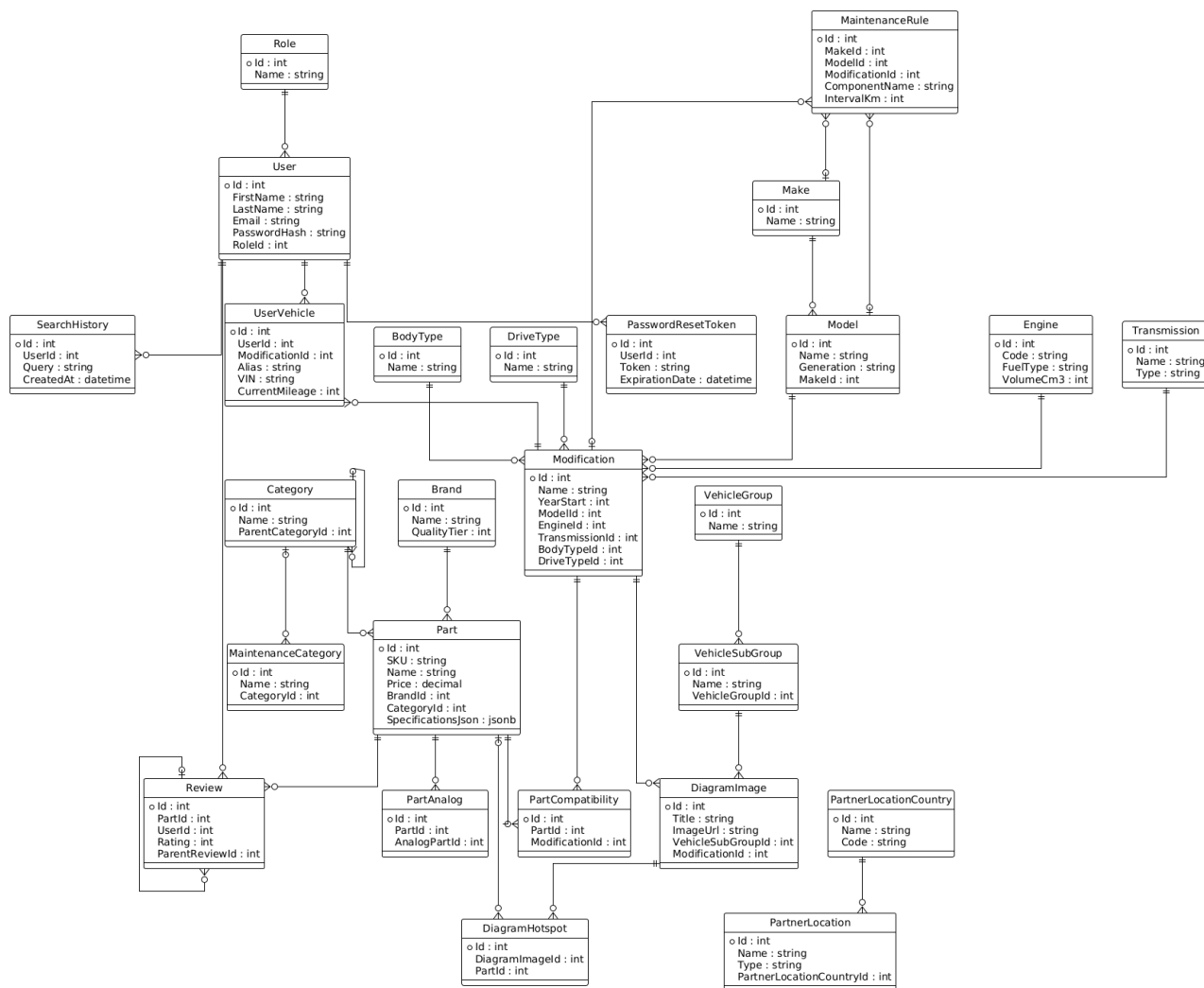


Рис. 2.2 – ER-діаграма таблиць та зв'язків бази даних

Модель бази даних наочно демонструє розгалужену реляційну структуру зберігання інформації в системі. Архітектуру бази даних спроектовано з дотриманням принципів нормалізації, що мінімізує надмірність даних та гарантує їх посилальну цілісність через використання системи первинних і зовнішніх ключів. Як видно з діаграми, усі сутності логічно згруповані у кілька ключових доменів. Центральне місце займають таблиці каталогу запчастин та багаторівневого довідника транспортних засобів, які взаємодіють між собою через проміжну таблицю сумісності. Це дозволяє ефективно реалізувати складний зв'язок типу «багато-до-багатьох», де одна деталь може підходити до різних модифікацій автомобілів, а одна модифікація включає тисячі сумісних деталей. Окрім базового каталогу, архітектура передбачає виокремлені підсистеми управління профілями користувачів, віртуальним гаражем,

геоданими партнерів, інтерактивними діаграмами та регламентами технічного обслуговування.

Для розуміння базових принципів функціонування платформи наведено опис найбільш значущих таблиць, що формують ядро бізнес-логіки розробленої системи.

Таблиця 2.4

Опис таблиці User

Поле	Тип	Опис
Id	int (PK)	Унікальний ідентифікатор
FirstName	string	Ім'я користувача
LastName	string	Прізвище користувача
Email	string (unique)	Електронна пошта
PhoneNumber	string?	Номер телефону
City	string?	Місто проживання
PasswordHash	string	Хеш паролю (BCrypt)
CreatedAt	DateTime	Дата реєстрації
RoleId	int (FK)	Зовнішній ключ до ролі

User — основна таблиця користувачів, яка має зв'язок «багато-до-одного» з таблицею Role для визначення рівня доступу (Customer, Moderator, Admin). Має зв'язок «один-до-багатьох» з таблицею UserVehicle (Віртуальний гараж) та з таблицею Review (відгуки).

Таблиця 2.5

Опис таблиці Part

Поле	Тип	Опис
Id	int (PK)	Унікальний ідентифікатор
SKU	string	Артикул виробника
Name	string	Назва запчастини
Description	string?	Текстовий опис

Price	decimal?	Ціна
ImageUrl	string?	URL зображення (Cloudinary)
WeightKg	double?	Вага (кг)
LengthMm, WidthMm, HeightMm	double?	Габарити (мм)
OemNumbers	List<string>	Масив OEM-номерів (PostgreSQL array)
EanCodes	List<string>	Масив EAN-кодів (штрих-коди)
SpecificationsJson	string?	Гнучкі характеристики (JSONB)
BrandId	int (FK)	Зовнішній ключ до бренду
CategoryId	int (FK)	Зовнішній ключ до категорії

Part — центральна таблиця каталогу. Використовує PostgreSQL native arrays для OEM та EAN, а також JSONB для зберігання гнучких технічних характеристик запчастини.

Таблиця 2.6

Опис таблиці PartAnalog

Поле	Тип	Опис
Id	int (PK)	Унікальний ідентифікатор
PartId	int (FK)	Оригінальна запчастина
AnalogPartId	int (FK)	Запчастина-аналог
MatchQuality	string?	Якість збігу (OEM Match, API Cross-Reference)

PartAnalog — самореферентна проміжна таблиця для зв'язку «багато-до-багатьох» між запчастинами з додатковим навантаженням (MatchQuality). Дозволяє будувати мережу прямих та зворотних крос-номерів.

Опис таблиці PartCompatibility

Поле	Тип	Опис
Id	int (PK)	Унікальний ідентифікатор
PartId	int (FK)	Зовнішній ключ до запчастини
ModificationId	int (FK)	Зовнішній ключ до модифікації

PartCompatibility — критично важлива проміжна таблиця, що реалізує зв'язок «багато-до-багатьох» між Part та Modification. Використовується для роботи алгоритмів підбору деталей під конкретне авто.

Опис таблиці Modification

Поле	Тип	Опис
Id	int (PK)	Унікальний ідентифікатор
Name	string	Назва модифікації
YearStart	int	Рік початку випуску
YearEnd	int?	Рік закінчення випуску
ModelId	int (FK)	Зовнішній ключ до моделі
EngineId	int (FK)	Зовнішній ключ до двигуна
TransmissionId	int (FK)	Зовнішній ключ до трансмісії
BodyTypeId	int (FK)	Зовнішній ключ до типу кузова
DriveTypeId	int (FK)	Зовнішній ключ до типу приводу

Modification — найдетальніший рівень довідника автомобілів із п'ятьма зовнішніми ключами. До цієї таблиці прив'язується сумісність запчастин та автомобілі у «Віртуальному гаражі». Об'єднує в собі посилання на модель, двигун, трансмісію, типи приводу та кузова. Саме до модифікації прив'язується сумісність запчастин.

Опис таблиці UserVehicle

Поле	Тип	Опис
Id	int (PK)	Унікальний ідентифікатор
UserId	int (FK)	Зовнішній ключ до користувача
ModificationId	int (FK)	Зовнішній ключ до модифікації
Alias	string	Персональне ім'я автомобіля
VIN	string?	VIN-код (17 символів)
CurrentMileage	int?	Поточний пробіг (км)

UserVehicle — реалізує функціонал збереження транспортних засобів, прив'язкою користувача до модифікації автомобіля, із підтримкою VIN-коду та пробігу для помічника ТО.

Таблиця 2.10

Опис таблиці MaintenanceRule

Поле	Тип	Опис
Id	int (PK)	Унікальний ідентифікатор
MakeId	int? (FK)	Зовнішній ключ до марки (nullable для загальних правил)
ModelId	int? (FK)	Зовнішній ключ до моделі
ModificationId	int? (FK)	Зовнішній ключ до модифікації
ComponentName	string	Назва компонента (Моторне масло, Фільтр повітряний тощо)
IntervalKm	int?	Інтервал заміни (км)
IntervalMonths	int?	Інтервал заміни (місяці)
Recommendation	string?	Текстова рекомендація

MaintenanceRule — визначає компоненти та інтервали їх заміни; підтримує каскадну специфічність через nullable FK: якщо всі FK = null — загальне правило;

якщо MakeId задано — для конкретної марки; аналогічно для моделі та модифікації.

Таблиця 2.11

Опис таблиці DiagramHotspot

Поле	Тип	Опис
Id	int (PK)	Унікальний ідентифікатор
ReferenceNumber	int	Номер позиції на схемі
PositionXPercent, PositionYPercent	double	Координати (%)
WidthPercent, HeightPercent	double	Розміри зони (%)
Label	string	Підпис
OemNumber	string?	ОЕМ-номер деталі
Quantity	int	Кількість штук
Supplement	string?	Додаткова інформація
PartId	int? (FK)	Пряма прив'язка до запчастини
DiagramImageId	int (FK)	Зв'язок з діаграмою

DiagramHotspot — пов'язує конкретне завантажене SVG-зображення (DiagramImageId) із конкретною запчастиною в каталозі (PartId), забезпечуючи візуальний пошук.

Таким чином, спроектована реляційна структура бази даних PostgreSQL є повністю нормалізованою, позбавленою інформаційної надмірності та оптимізованою для читання. Завдяки правильному використанню зовнішніх ключів, база даних створює надійний, консистентний фундамент для роботи інформаційної системи, забезпечуючи високу швидкість каталогу та безпомилкову роботу алгоритмів крос-пошуку автозапчастин.

2.3. Проектування та реалізація алгоритмів системи

Алгоритми мають ключове значення для забезпечення ефективної та коректної роботи інформаційної системи. Для моделювання алгоритмів роботи системи та відображення основних етапів взаємодії користувача з платформою підбору автозапчастин були створені діаграми активності [21].

Користувач починає взаємодію з системою, обираючи одну з трьох доступних дій — авторизацію через email/password, авторизацію через Google OAuth або реєстрацію. При авторизації через email система перевіряє хеш пароля BCrypt у базі даних; при Google OAuth — валідує ID-токен Google та автоматично створює акаунт при першому вході. У разі успішної автентифікації генерується JWT-токен із claims (NameIdentifier, Email, Role, FullName) та терміном дії 24 години (рис. 2.1).

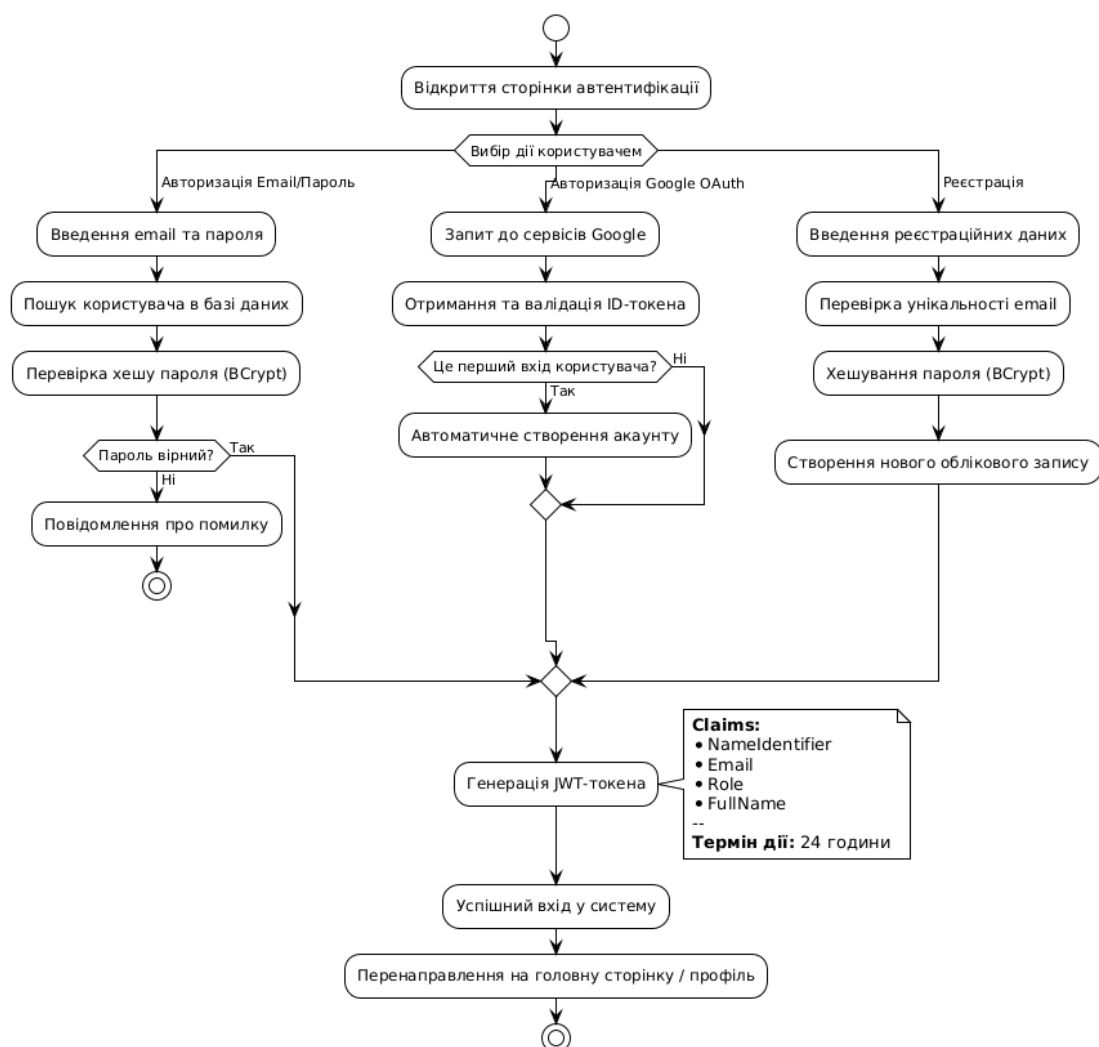


Рис. 2.3 – Діаграма активності при першому вході на сайт

Процес підбору запчастин (рис. 2.4) починається з двох основних шляхів: прямий пошук через Elasticsearch або підбір через «Віртуальний гараж». При прямому пошуку система виконує MultiMatch-запит до Elasticsearch з нечітким зіставленням та зваженим ранжуванням [6, 17]. При підборі через гараж — користувач обирає марку/модель/модифікацію або використовує VIN-

На сторінці запчастини доступні: перевірка сумісності з авто з гаража, перегляд аналогів з ValueScore, перегляд та залишення відгуків, перегляд на діаграмі вузла (якщо запчастина прив'язана до DiagramHotspot).



розраховуються кілометраж попереднього та наступного планового сервісного обслуговування. Якщо до планової заміни залишається не більше двох тисяч кілометрів, або пробіг перевищено не більше ніж на тисячу кілометрів, система автоматично ідентифікує потребу в обслуговуванні. На основі цих даних формується відповідний перелік рекомендацій у форматі HTML-листа з прямими посиланнями на каталог деталей, який автоматично надсилається користувачу електронною поштою.

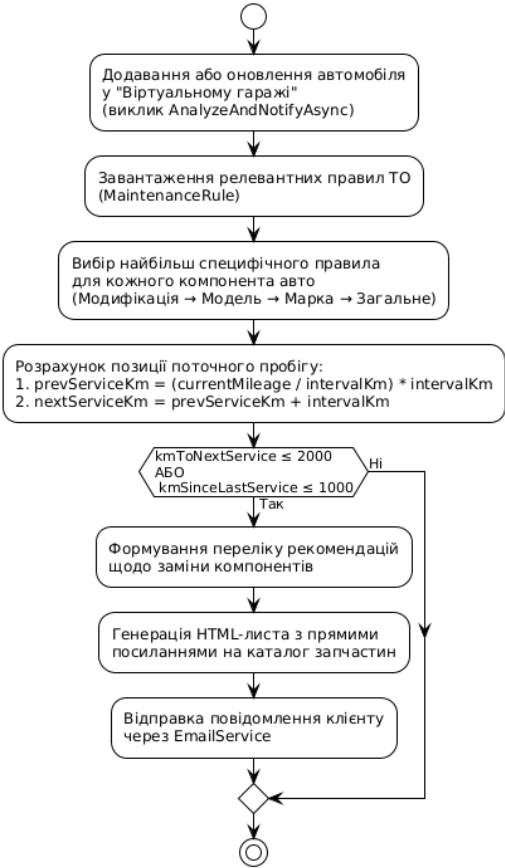


Рис. 2.5 – Діаграма активності алгоритму помічника технічного обслуговування

Для об'єктивної оцінки взаємозамінних деталей та допомоги користувачу у виборі реалізовано алгоритм розрахунку метрики ValueScore (рис. 2.6). Оцінка найкращого співвідношення ціни та якості розраховується за спеціальною формулою, що враховує рівень якості бренду та ефективну ціну запчастини. Для оригінальних деталей застосовується коригувальний коефіцієнт, який підвищує значущість рівня якості. Ефективна ціна визначається як фактична вартість товару, а у разі її відсутності використовується резервний механізм —

обчислюється середня ціна в межах аналогічного рівня якості або береться глобальна середня ціна. Після проведення розрахунків для всіх знайдених позицій, аналог із найвищим показником ValueScore та наявною фактичною ціною отримує спеціальну позначку найкращого вибору, що візуально виділяє його в інтерфейсі каталогу.

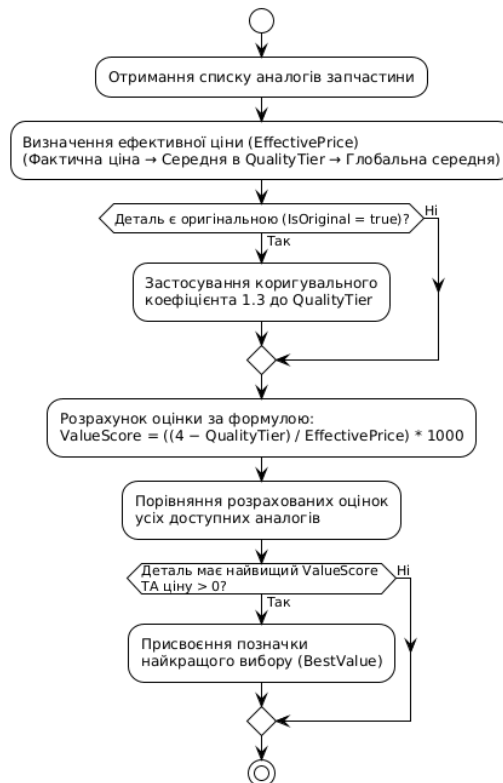


Рисунок 2.6 – Діаграма активності алгоритму розрахунку метрики ValueScore

У процесі проєктування алгоритмів пошуку постало критичне завдання забезпечення суворої узгодженості (консистентності) даних між основним сховищем PostgreSQL та пошуковим рушієм Elasticsearch. Для вирішення цієї проблеми було розроблено алгоритм подвійного запису із застосуванням патерну «Спостерігач» на рівні доступу до даних. Відповідно до принципів Clean Architecture, взаємодію з пошуковим рушієм абстраговано через інтерфейс, який інкапсулює методи індексування та видалення. Інтеграція цього алгоритму здійснюється безпосередньо в життєвий цикл бази даних: під час виклику збереження змін серверна система перехоплює транзакцію. Якщо адміністратор додає або оновлює запчастину в PostgreSQL, алгоритм автоматично формує

спеціальний об'єкт передачі даних, який містить лише індексовані поля (артикул, назва, крос-номери), і асинхронно відправляє його до Elasticsearch. Натомість, у разі видалення товару з каталогу, викликається метод з передачею ідентифікатора деталі. Це гарантує, що видалена позиція негайно вилучається з пошукової видачі рушія, що унеможливлює появу «фантомних» результатів, коли деталь відображається в пошуку, але її сторінка вже не існує, та забезпечує абсолютну точність роботи платформи [30].

2.4. Створення функціоналу системи

Перед початком безпосереднього кодування програмного коду було проведено комплексний етап проєктування реляційної моделі бази даних. Використання графічних інструментів моделювання дозволило чітко визначити структури двадцяти п'яти таблиць та налаштувати зв'язки між ключовими сутностями. Особливістю реалізації стало використання специфічних типів даних PostgreSQL, зокрема нативних масивів для зберігання списків OEM-номерів та формату JSONB для збереження динамічних технічних характеристик деталей. Такий підхід дозволив уникнути створення надмірної кількості нормалізованих таблиць для дрібних атрибутів товарів, зберігши при цьому високу швидкість виконання запитів [5, с. 58].

На основі спроектованої моделі реалізовано кодову структуру серверної частини за принципами Clean Architecture, що гарантує незалежність бізнес-логіки від зовнішніх фреймворків [4]. Кожна таблиця бази даних представлена окремим класом-сутністю на мові C# у доменному шарі. Для зручності підтримки коду класи логічно згруповані за предметними областями: модуль каталогу (запчастини, бренди, категорії, аналоги), модуль транспортних засобів (марки, моделі, модифікації, типи двигунів), модуль користувачів та модуль управління діаграмами. Централізована конфігурація всіх колекцій даних реалізована в інфраструктурному шарі через єдиний контекст бази даних. За допомогою Fluent API в Entity Framework Core налаштовано правила каскадного видалення, обмеження довжини рядків, унікальні індекси для артикулів та мапінг

складних типів [15]. Обробка бізнес-логіки винесена у шар Application, де реалізовано сервіси з використанням патерну Dependency Injection.

Для забезпечення ідентичності середовищ розробки та продакшну інфраструктуру було розгорнуто з використанням технології контейнеризації Docker. Файл конфігурації визначає запуск пошукового рушія Elasticsearch із налаштуванням збереження індексів на постійному томі (persistent volume), що запобігає втраті даних після перезавантаження сервера. З огляду на високі вимоги до швидкодії відображення медіаконтенту, зберігання зображень було винесено за межі локального сервера [11, с. 90]. Було розроблено сервіс, який реалізує інтерфейс завантаження файлів до хмарної платформи Cloudinary. Цей сервіс забезпечує не лише збереження фотографій, але й їхню автоматичну конвертацію у сучасний формат WebP та генерацію оптимізованих URL-адрес для миттєвої доставки клієнтам через CDN [29].

Для захисту API в класичному сценарії використано JSON Web Token (JWT) з криптографічним підписом HMAC-SHA256. Токен містить базові ідентифікатори (claims): ідентифікатор користувача, електронну пошту, роль у системі (Admin, Customer) та повне ім'я. На клієнтській стороні реалізовано HTTP-перехоплювачі через бібліотеку Axios, які автоматично додають токен із локального сховища до заголовка кожного мережевого запиту. Глобальний контекст авторизації забезпечує доступ до стану користувача через React Context API, включаючи функціонал автоматичного парсингу JWT для коректного відображення елементів інтерфейсу, наприклад, приховування адмін-панелі для звичайних користувачів.

Функціональну реалізацію клієнтської та серверної частин розпочато зі створення модуля авторизації та безпеки [9, с. 92]. Враховуючи специфіку сучасних вебзастосунків, окрім класичної реєстрації через електронну пошту, було імплементовано механізм швидкого входу (SSO) за допомогою Google OAuth 2.0. Інтерактивну взаємодію компонентів системи під час авторизації через сторонні сервіси наочно проілюстровано на діаграмі послідовності (рис. 2.7).

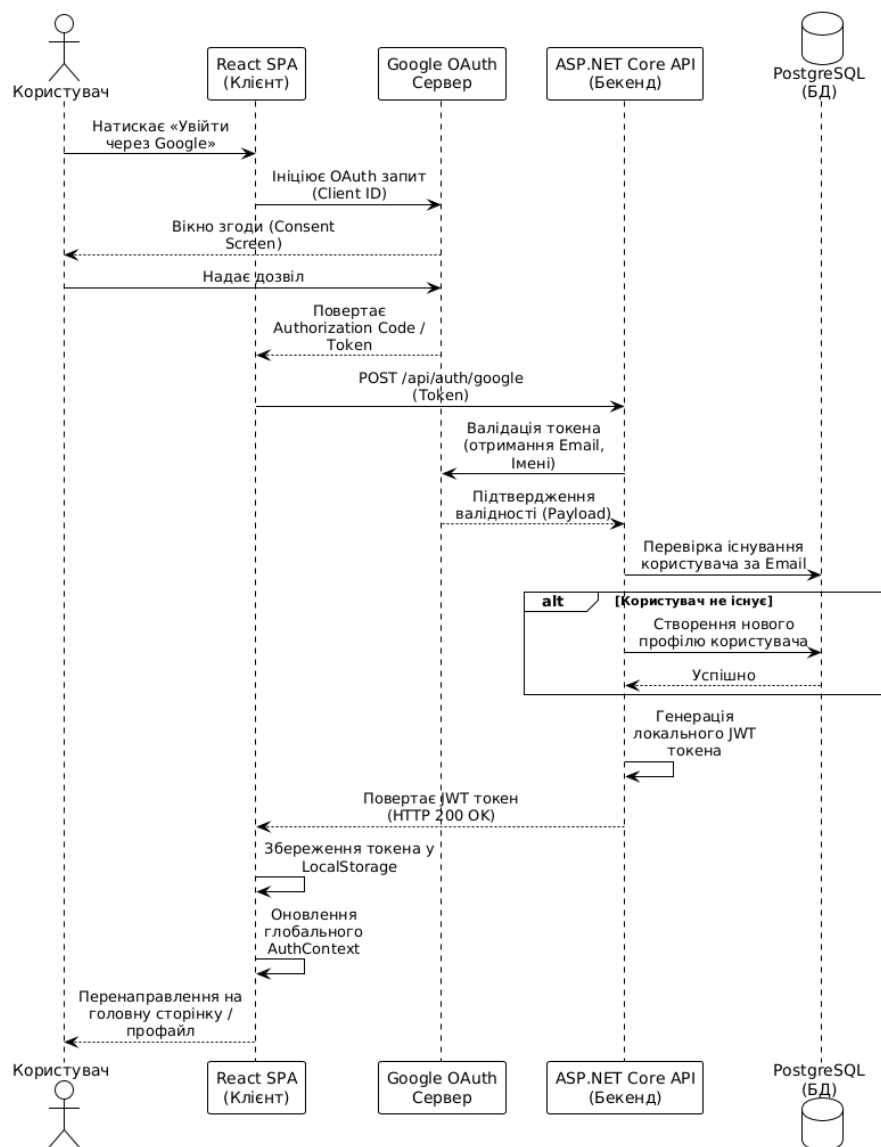


Рис. 2.7 – Діаграма послідовності процесу авторизації через Google OAuth

Центральним ядром системи є функціонал каталогу та модулів пошуку [6].

Реалізовано двоканальну архітектуру обробки запитів: швидкі підказки (suggest) в реальному часі обробляються базою даних через оптимізовані SQL-запити з ранжуванням за збігом початку рядка, тоді як складний повнотекстовий пошук перенаправляється до виділеного сервісу Elasticsearch. Це дозволяє знаходити деталі навіть при допущенні друкарських помилок або неправильній розкладці клавіатури завдяки алгоритмам нечіткого зіставлення. Компонент вибору автомобіля (VehicleSelector) на клієнтській частині реалізує логіку каскадної вибірки: вибір марки розблоковує список моделей, що, своєю чергою, завантажує відповідні модифікації. Механізм пошуку аналогів реалізує складний

алгоритм двонаправленого пошуку та розрахунку метрики ValueScore, яка враховує відсоток збігу характеристик та рейтинг бренду заміни.

Для забезпечення швидкодії клієнтської частини (фронтенду) впроваджено власну логіку кешування та дедуплікації мережових запитів. Компонент історії пошуку забезпечує гібридне зберігання даних: для неавторизованих відвідувачів історія зберігається локально у браузері, а після входу в систему автоматично синхронізується із сервером. Особливої уваги заслуговує реалізація модуля інтерактивних діаграм: система дозволяє адміністратору не лише завантажувати зображення вузлів, але й розраховувати відносні координати розташування деталей на схемі, що забезпечує коректне відображення клікабельних зон незалежно від розміру екрана пристрою користувача.

Уся розроблена бізнес-логіка супроводжується суворою валідацією даних на обох архітектурних рівнях. Серверна перевірка реалізована безпосередньо у контролерах та сервісах: перевіряється унікальність артикулів, існування зв'язків за зовнішніми ключами, а також права доступу користувачів до виконання певних операцій. Клієнтська валідація, побудована на базі бібліотек роботи з формами, надає миттєвий зворотний зв'язок користувачеві та унеможливорює відправку некоректних або неповних даних до сервера [13]. У підсумку, використання сучасного технологічного стеку .NET 8 та React зі збірником Vite дозволило створити надійну, продуктивну та гнучку інформаційну систему з повноцінною інтеграцією розширених функціональних можливостей.

Висновки до другого розділу

У другому розділі проаналізовано вимоги до системи, визначено ролі користувачів і ключовий функціонал: алгоритми VIN-декодування, помічника технічного обслуговування, інтерактивні діаграми та геопошук. Архітектуру спроектовано для швидкої обробки великих масивів технічних даних, а логіку базових бізнес-процесів (від авторизації до розрахунку ValueScore) візуалізовано за допомогою діаграм активностей.

Серверну частину побудовано за принципами Clean Architecture, що гарантує високу модульність коду. Спроектовано структуру бази даних

					ІІЗ.КР.Б – 121 – 26 - ІІЗ	Арк.
						45

PostgreSQL із двадцяти п'яти таблиць, яка забезпечує цілісність ієрархії транспортних засобів і гнучке збереження специфікацій завдяки нативним типам масивів та JSONB.

Безпековий контур реалізовано через JWT-авторизацію, Google OAuth та хешування BCrypt. Технічну базу підсилено інтеграціями з рушієм Elasticsearch, хмарним сховищем Cloudinary та зовнішніми API. Отже, результати розділу формують стійке підґрунтя для практичної реалізації безпечної та високопродуктивної платформи.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ФУНКЦІОНАЛЬНИХ МОДУЛІВ СИСТЕМИ

3.1. Порядок встановлення та налаштування параметрів системи

Розгортання платформи AutoParts є комплексним процесом, що охоплює підготовку серверного середовища, налаштування бази даних, запуск інфраструктурних сервісів у контейнерному середовищі, конфігурування серверної та клієнтської частин застосунку.

Завдяки застосуванню технологій контейнеризації та чіткому поділу застосунку на шари процес розгортання є відтворюваним та незалежним від конкретної операційної системи [19]. Успішна інсталяція вимагає дотримання встановлених вимог до апаратного та програмного середовища (табл. 3.1).

Таблиця 3.1

Вимоги до середовища розгортання системи AutoParts

Компонент	Мінімальна вимога	Рекомендована конфігурація
Процесор	2 ядра	4 ядра і більше
Оперативна пам'ять	4 ГБ	8 ГБ (враховуючи потреби Elasticsearch — 512 МБ JVM heap)
Дисковий простір	10 ГБ	20 ГБ і більше
Операційна система	Windows 10, Ubuntu 20.04, macOS 12	Ubuntu 22.04 LTS
Платформа .NET	.NET SDK 8.0	.NET SDK 8.0.x (LTS)
Середовище виконання JS	Node.js 20.x	Node.js 20.x LTS + npm 10
СУБД	PostgreSQL 15	PostgreSQL 15+
Контейнеризація	Docker Desktop 4.x	Docker Engine 24+ + docker-compose
Система контролю версій	Git 2.40+	Git 2.40+

Серверну частину системи організовано за принципами Clean Architecture та структуровано у вигляді рішення .NET 8, що складається з чотирьох проєктів: Domain (доменні сутності та бізнес-правила), Application (інтерфейси сервісів та об'єкти передачі даних), AutoPartsSystem.Infrastructure (реалізація доступу до

бази даних та зовнішніх інтеграцій) і AutoPartsSystem.API (вебконтролери та загальні налаштування). Клієнтську розробку винесено в окрему директорію AutoPartsFrontend як односторінковий застосунок (SPA) на базі React та збірника Vite.

Базовим етапом розгортання платформи є запуск пошукового рушія Elasticsearch 8.12 в ізольованому Docker-контейнері згідно з конфігурацією docker-compose.yml [26]. Сервіс налаштовано в режимі одного вузла (single-node) з вимкненою XPack-аутентифікацією та обмеженням пам'яті JVM до 512 МБ, що гарантує його стабільну роботу навіть у середовищах з обмеженими ресурсами. Збереження пошукових індексів між перезапусками контейнера забезпечується монтуванням іменованого тому es_data. Після ініціалізації рушій готовий приймати запити від бекенду на порту 9200. Управління всіма параметрами підключення та секретними ключами інтеграцій здійснюється централізовано через конфігураційний файл appsettings.json, загальна структура якого налічує шість основних секцій (табл. 3.2).

Таблиця 3.2

Секції конфігурації серверного застосунку

Секція	Призначення	Ключові параметри
ConnectionStrings	Підключення до PostgreSQL 15	Host, Port, Database, Username, Password
Jwt	Параметри JWT-авторизації	Key (секретний ключ HMAC-SHA256), Issuer, Audience
Smtp	SMTP-сервер для email-повідомлень	Host, Port, Username, Password (Mailtrap у розробці)
CloudinarySettings	Хмарне CDN-сховище зображень	CloudName, ApiKey, ApiSecret
BMW Parts Catalog API	Автозаповнення даних запчастини	ApiKey, Host, BaseUrl
Cors	CORS-політика безпеки	AllowedOrigins (Vercel URL + localhost)

Після завершення налаштування конфігурації ініціалізація схеми реляційної бази даних виконується за допомогою механізму автоматичних міграцій Entity Framework Core. Утиліта командного рядка dotnet-ef аналізує поточний стан сховища та послідовно застосовує необхідні зміни [23], формуючи в середовищі PostgreSQL повну структуру з двадцяти п'яти таблиць. Завдяки ідемпотентності цього процесу повторний запуск міграцій є безпечним і гарантує збереження вже існуючих даних.

Паралельно, розгортання клієнтської частини вимагає налаштування змінних середовища у файлі .env.local, де визначаються базова URL-адреса серверного програмного інтерфейсу та ідентифікатор клієнта для коректної роботи модуля авторизації Google OAuth 2.0. Загальна топологія розгортання системи, що наочно демонструє архітектурну взаємодію між клієнтським односторінковим застосунком, серверним API, базами даних та зовнішніми сервісами, проілюстрована на рисунку 3.1.

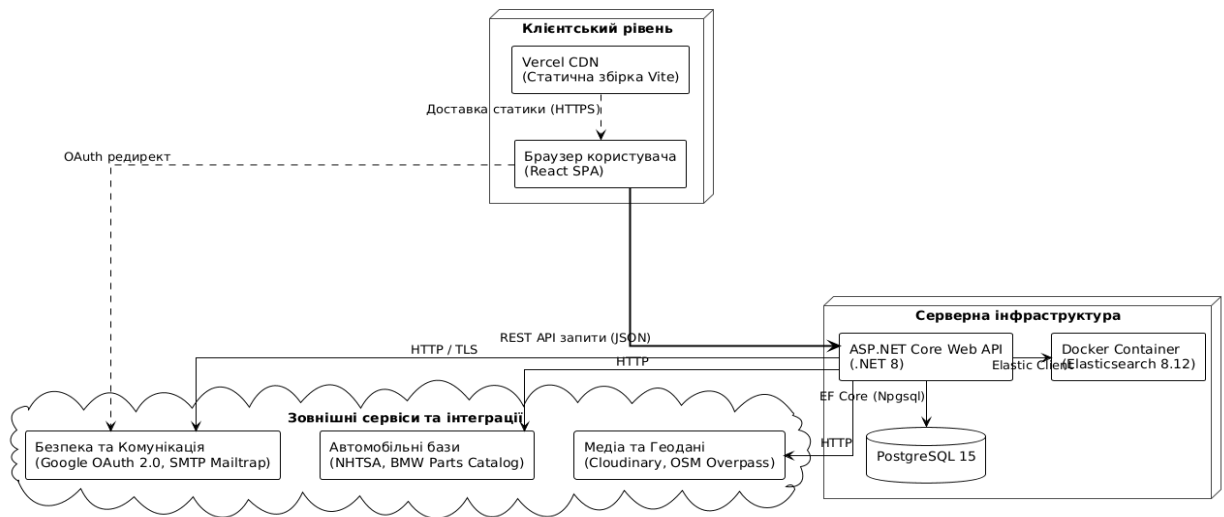


Рис. 3.1 – Діаграма розгортання системи AutoParts

Організацію процесу розробки системи побудовано на засадах методології DevOps та використання автоматизованих конвеєрів безперервної інтеграції. Для управління версіями вихідного коду застосовується система Git із стратегією розгалуження Feature Branch Workflow. Ця стратегія передбачає наявність трьох рівнів: стабільної гілки main, що відповідає актуальному задокументованому релізу, інтеграційної гілки develop для злиття завершеного функціоналу перед

виходом у продакшн, а також набору ізольованих гілок feature/*, які гарантують безпечну розробку модулів без ризику дестабілізації основної кодової бази.

Конвеєр безперервної інтеграції та доставки реалізовано за допомогою інструментарію GitHub Actions. Його конфігурацію описано у відповідному YAML-файлі, а запуск відбувається автоматично під час кожного оновлення коду (push) або створення запитів на злиття (pull request) до головних гілок. Процес виконання розділено на два паралельні завдання (табл. 3.3).

Таблиця 3.3

Склад завдань конвеєра CI/CD (GitHub Actions)

Назва завдання	Середовище виконання	Послідовність кроків
Frontend Lint & Security	ubuntu-latest, Node.js 20	Встановлення залежностей (npm ci); статичний аналіз коду через ESLint (npm run lint); перевірка безпеки залежностей (npm audit --audit-level=high).
Backend Build, Test & SonarCloud	ubuntu-latest, .NET 8, Java 17, PostgreSQL 15 (як сервіс)	Перевірка форматування коду; перевірка вразливостей NuGet-пакетів; встановлення dotnet-ef та застосування міграцій до тестової БД; збірка проєкту (dotnet build); запуск тестів зі збором метрик покриття; статичний аналіз якості коду SonarCloud; завантаження артефакту зі звітом про покриття.

Важливою архітектурною перевагою налаштованого бекенд-конвеєра є запуск повноцінної бази даних PostgreSQL як фонового сервісу всередині середовища GitHub Actions. Це дозволяє виконувати не лише ізольовані модульні тести в оперативній пам'яті, а й комплексні інтеграційні тести, що потребують реального підключення до сховища даних. Загальну логіку роботи автоматизованого конвеєра проілюстровано на рисунку 3.2.

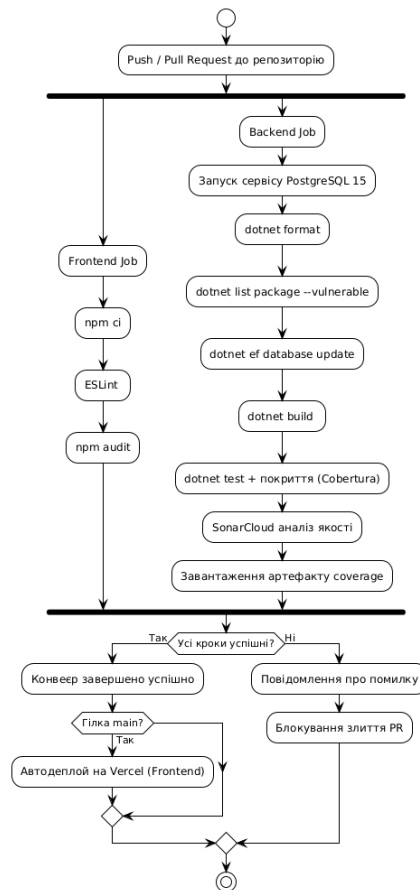


Рис. 3.2 – Схема роботи конвеєра CI/CD

Для оперативного моніторингу стану розгорнутої системи та її інфраструктурних залежностей у серверну частину на базі ASP.NET Core було інтегровано механізм Health Checks (перевірка працездатності). Цей інструментарій дозволяє автоматизованим системам та адміністраторам швидко отримувати інформацію про доступність платформи.

Після завершення всіх етапів конфігурування, успішного запуску Docker-контейнерів із базою даних PostgreSQL та пошуковим рушієм Elasticsearch, а також ініціалізації самого вебсервера, система стає доступною для перевірки [26]. Звернення до спеціального системного ендпоінту /health повертає статус Healthy (рис. 3.3), що підтверджує коректну роботу ядра застосунку та його готовність приймати HTTP-запити від клієнтського SPA-інтерфейсу.

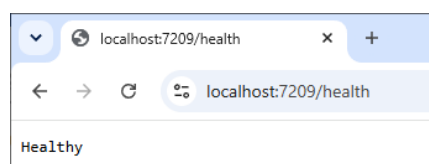


Рис. 3.3 – Успішна перевірка працездатності серверного API

3.2. Структура інтерфейсу та порядок взаємодії з платформою

Клієнтську частину платформи реалізовано як односторінковий застосунок (SPA) за допомогою бібліотеки React 18, збірника Vite та маршрутизатора React Router v6 [24, 25]. Візуальне оформлення інтерфейсу витримано у єдиній сучасній світлій кольоровій гамі з домінуючим темно-синім фоном, яскравими акцентами для карток, модальних вікон та оверлеїв. Для підвищення інтерактивності та створення відчуття плавності взаємодії широко застосовуються CSS-мікроанімації переходів. Побудований за принципами адаптивного дизайну, застосунок гарантує коректне та зручне відображення на екранах будь-якого розміру: зокрема, на мобільних пристроях стандартна горизонтальна панель навігації автоматично трансформується у бічне висувне меню (drawer) з елементом керування типу «гамбургер».

Загальну навігаційну структуру системи логічно розподілено на три рівні доступу залежно від статусу та ролі відвідувача. Детальний перелік маршрутів вебзастосунку та відповідні права доступу наведено у таблиці нижче.

Таблиця 3.4

Навігаційна структура та рівні доступу до маршрутів системи

Маршрут	Назва сторінки	Рівень доступу
/	Головна сторінка	Публічний
/catalog	Каталог запчастин	Публічний
/parts/:id	Детальна картка запчастини	Публічний
/search	Результати повнотекстового пошуку	Публічний
/diagrams	Інтерактивні діаграми вузлів	Публічний
/partner-locations	Пошук автомагазинів та сервісів	Публічний
/login, /register	Авторизація та реєстрація	Публічний
/garage	Віртуальний гараж	Авторизований користувач
/maintenance	Помічник технічного обслуговування	Авторизований користувач
/profile	Особистий профіль	Авторизований користувач
/search-history	Історія пошуку	Авторизований користувач
/admin/dashboard	Аналітичний дашборд	Адміністратор
/admin/parts	Управління каталогом запчастин	Адміністратор
/admin/brands	Управління брендами	Адміністратор
/admin/categories	Управління категоріями	Адміністратор
/admin/vehicles	Управління довідником автомобілів	Адміністратор
/admin/diagrams	Управління діаграмами вузлів	Адміністратор

/admin/maintenance-rules	Управління регламентами ТО	Адміністратор
/admin/partner-locations	Управління партнерськими локаціями	Адміністратор
/admin/users	Управління користувачами та ролями	Адміністратор

Захист маршрутів реалізовано на рівні React Router через компонент-обгортку, який при спробі доступу до захищеного маршруту перевіряє наявність та валідність JWT-токена у локальному сховищі браузера. За відсутності токена виконується перенаправлення на сторінку авторизації. На рівні серверного API кожен захищений ендпоінт додатково анотовано атрибутом авторизації з перевіркою ролі, що унеможливорює несанкціонований доступ навіть при прямому HTTP-запиті.

Головна сторінка (рис. 3.4) є точкою входу в систему для будь-якого відвідувача. У верхній частині розміщено навігаційну панель із посиланнями на основні розділи: Каталог, Пошук, Схеми вузлів, Регламенти ТО, а також кнопкою «Увійти» для авторизації.

Центральним елементом сторінки є головний інформаційний блок (hero section) із заголовком та коротким описом платформи. Для швидкої навігації тут передбачено дві функціональні кнопки: «Переглянути каталог» та «Пошук за артикулом».

Нижче розташований інтерактивний блок «Точний підбір за автомобілем». Він реалізований у вигляді покрокового майстра (wizard), який дозволяє користувачеві послідовно обрати марку, модель, покоління, рік випуску, тип кузова та модифікацію транспортного засобу для точного фільтрування сумісних деталей. Наступна секція візуалізує ключові інструменти платформи у вигляді зручних карток із іконками. Серед них: пошук за номером, підбір за VIN-кодом, загальний каталог запчастин, інтерактивні схеми вузлів, регламенти ТО та блок доступу до персонального Гаража.

Рис. 3.5 – Сторінка авторизації користувачів

Рис. 3.6 – Сторінка реєстрації користувачів

Сторінка каталогу (рис. 3.7) організована за триколонковим адаптивним макетом. Ліворуч розміщено панель із деревом ієрархічних категорій: кожна категорія може містити підкатегорії будь-якої глибини вкладеності, що відображаються у вигляді розкритих вузлів дерева. При виборі категорії центральна колонка автоматично оновлює перелік запчастин, а права колонка — динамічно формує панель фільтрів на основі фактичних технічних характеристик запчастин у поточній категорії (spec-keys). Над переліком товарів розміщено компонент VehicleSelector — послідовний вибір «Марка → Модель → Модифікація», при активації якого каталог фільтрується за таблицею сумісності PartCompatibility. Кожна картка товару у центральній колонці містить фотографію з Cloudinary CDN, назву, бренд із кольоровою позначкою рівня якості, ціну та кнопку переходу до детальної картки. Сторінка підтримує пагінацію.

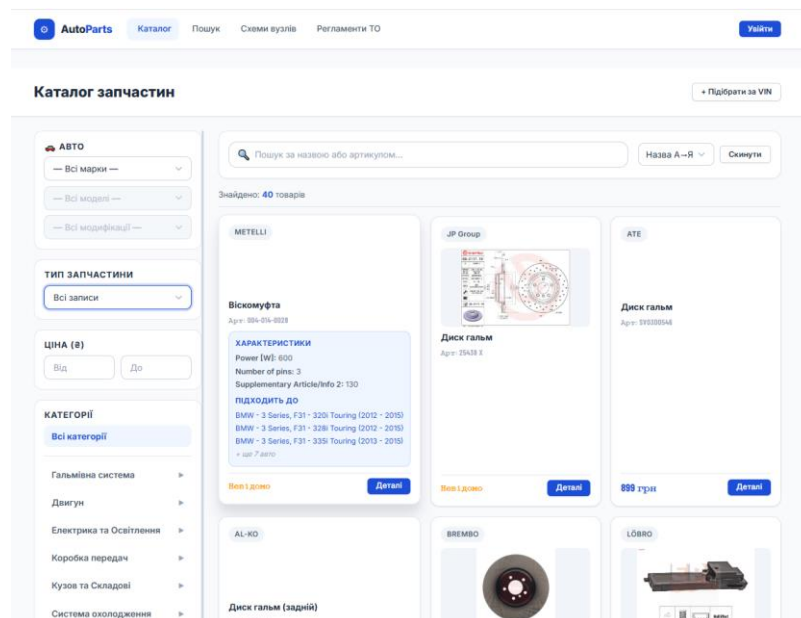


Рис. 3.7 – Каталог запчастин із деревом категорій та динамічними фільтрами

Сторінка пошуку (рис. 3.8) відображається при переході з головного пошукового рядка. Зверху відображається рядок пошуку із поточним запитом, що дозволяє одразу уточнити або змінити запит. Нижче — перелік знайдених запчастин, відсортованих за релевантністю. Пошук виконується через MultiMatch-запит Elasticsearch по шести полях: SKU (ваговий коефіцієнт 3.0), Name (2.0), BrandName, CategoryName, CrossNumbers та SpecificationsText (1.0 кожне) [30]. Алгоритм нечіткого зіставлення (Fuzziness AUTO) дозволяє знаходити релевантні результати навіть при незначних помилках у введеному запиті або частковому збігу артикулу [17]. Кожна картка результату містить ті самі ключові атрибути, що і в каталозі.

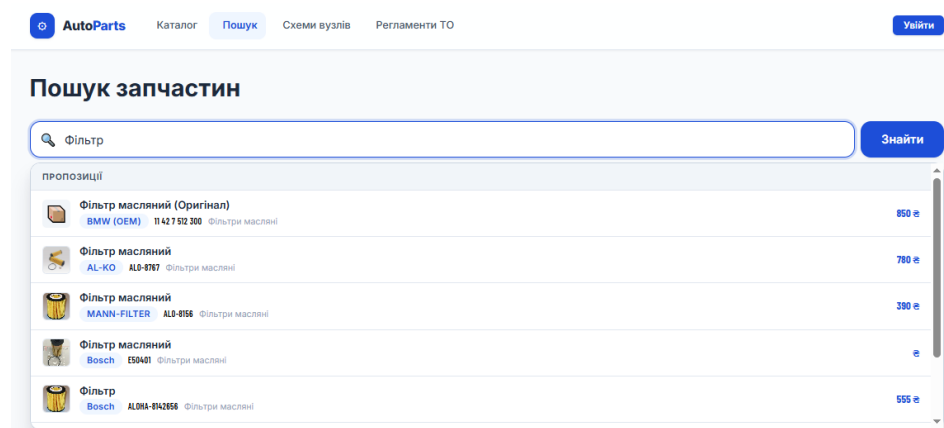


Рис. 3.8 – Сторінка результатів повнотекстового пошуку через Elasticsearch

Сторінка запчастини (рис. 3.9) має двоколонковий макет. Ліва колонка містить зображення запчастини, завантажене з Cloudinary CDN [29]. При відсутності зображення відображається нейтральний плейсхолдер. Права колонка містить основну інформацію про товар: назва, артикул (SKU) та ціну з можливістю копіювання, а також перевіркою на сумісність із автомобілем в гаражі. Нижче відображається таблиця технічних характеристик — вмісту поля SpecificationsJson, де ключі є назвами параметрів, а значення — їхніми величинами. Окремими блоками виводяться масиви OEM-номерів та EAN-кодів з можливістю копіювання кожного; список сумісних модифікацій автомобілів із зазначенням марки, моделі та років випуску.

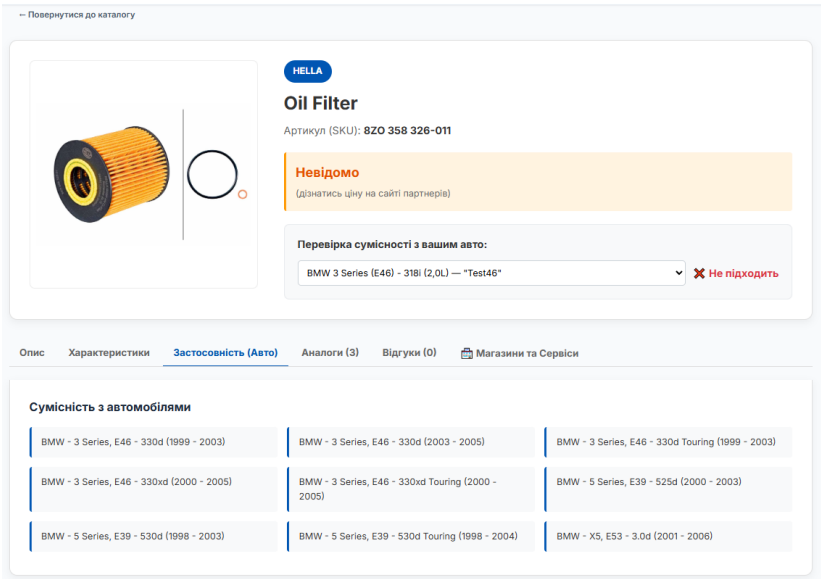


Рис. 3.9 – Детальна картка запчастини (основна інформація та сумісність)

Під основною інформацією розміщено панель із блоками-вкладками. Вкладка «Аналоги» (рис. 3.10) відображає перелік альтернативних запчастин, відсортованих за алгоритмом ValueScore від найвищого до найнижчого балу. Запчастина з максимальним показником ValueScore виділяється зеленою міткою «Найкраще співвідношення ціни та якості». Для кожного аналога відображається бал ValueScore, відповідність (MatchQuality) та посилання на його картку. Вкладка «Відгуки» (рис. 3.11) містить форму для авторизованих користувачів (вибір кількості зірок від 1 до 5, текстове поле для коментаря та кнопка збереження) та пронумерований перелік існуючих відгуків. Кожен відгук містить ім'я автора, рейтинг у вигляді зірок, дату публікації, текст та кнопку

«Відповісти», яка розгортає форму для ієрархічного коментаря. Вкладка «Магазини та сервіси» (рис. 3.12) відображає вбудовану інтерактивну карту для пошуку найближчих партнерів.

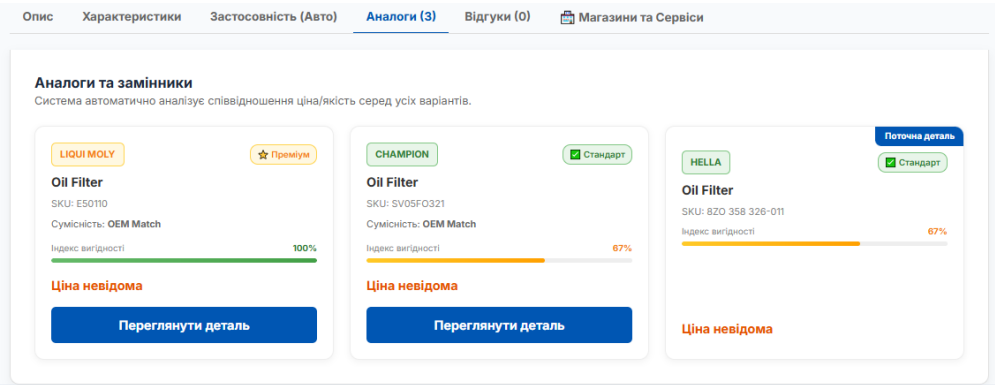


Рис. 3.10 – Вкладка «Аналоги» з відображенням алгоритму ValueScore

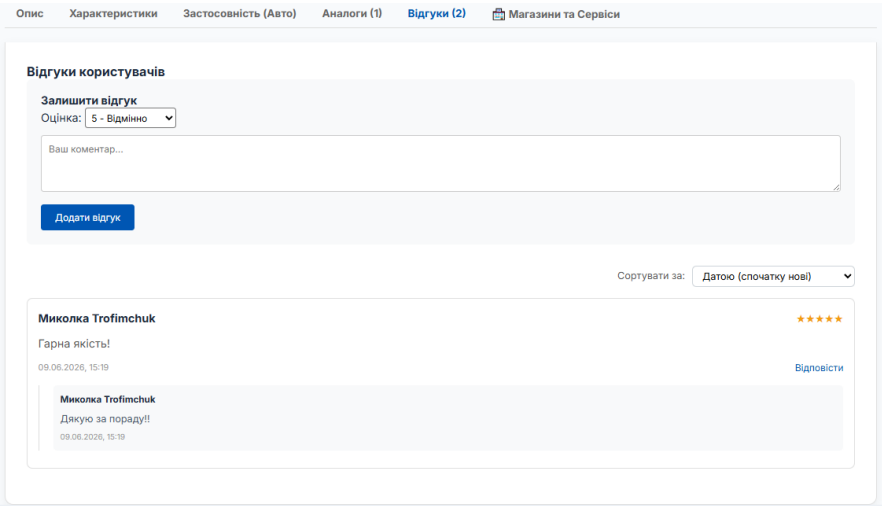


Рис. 3.11 – Вкладка «Відгуки» сторінки запчастини

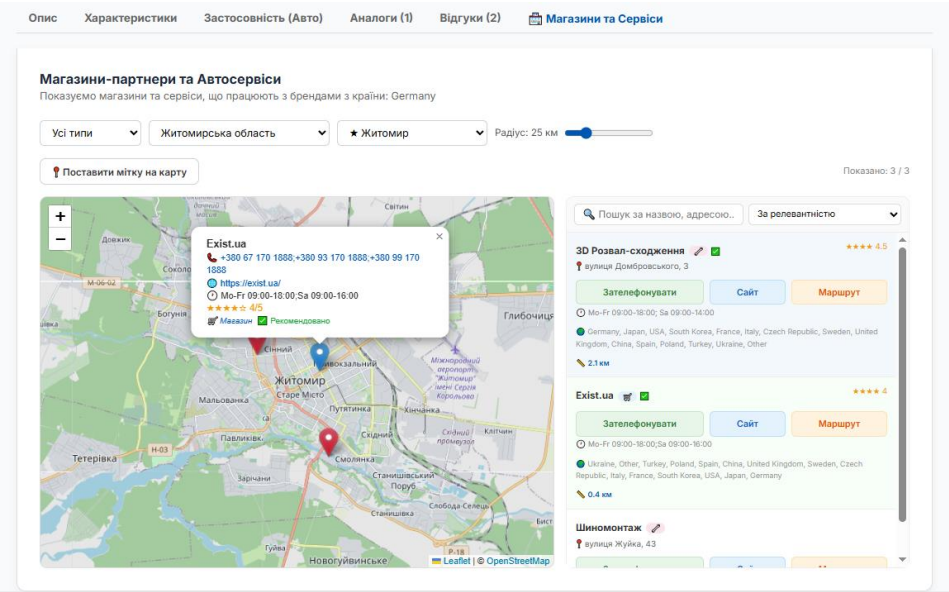


Рис. 3.12 – Вкладка «Партнерів» сторінки запчастини

Сторінка гаража /garage доступна лише авторизованим користувачам. Вона відображає перелік збережених автомобілів у вигляді горизонтальних карток (рис. 3.13). Кожна картка містить персональне ім'я автомобіля (alias), встановлене користувачем, повну назву (Марка / Модель / Рік випуску / Модифікація), VIN-код (якщо вказаний) та поточний пробіг у кілометрах. Для кожного авто доступні кнопки редагування (зміна alias, пробігу та VIN) та видалення. У верхньому правому куті сторінки розміщено кнопку «Додати автомобіль», яка відкриває модальне вікно з двома вкладками. Перша вкладка — ручний вибір (рис. 3.14) — надає послідовні спадні списки Марки, Моделі та Модифікації. Друга вкладка — «Декодувати VIN» — містить поле введення 17-символьного коду та кнопку запиту до NHTSA API (рис. 3.15): після отримання відповіді система відображає ідентифіковані дані (марка, модель, рік, двигун) та пропонує підтвердити їх для збереження [27]. Нижче поля VIN розміщено поле для введення поточного пробігу та текстового alias.

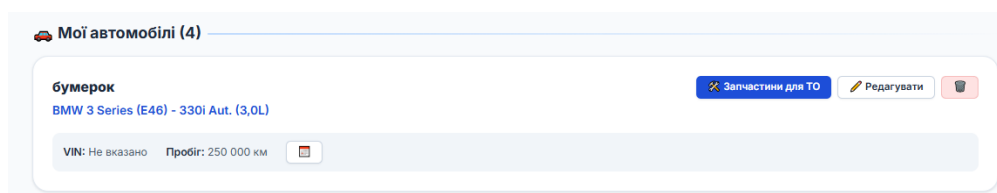


Рис. 3.13 – Сторінка «Віртуальний гараж» — збережені автомобілі

Рис. 3.14 – Сторінка «Віртуальний гараж» — ручне додавання автомобіля

Сторінка /diagrams (рис. 3.17) реалізує трирівневу навігацію по схемах вузлів автомобіля. Ліворуч розміщено панель вибору: спочатку обирається група вузлів, потім — підгрупа, і нарешті — конкретна діаграма із назвою та прив'язкою до модифікації автомобіля. У правій частині відображається SVG-схема обраного вузла в натуральну величину. Поверх SVG-зображення накладено інтерактивні точки (hotspot), кожна з яких розміщена за відсотковими координатами відносно розмірів зображення, що забезпечує коректне позиціонування при будь-якому масштабі. При наведенні курсора на точку з'являється tooltip із порядковим номером деталі, її назвою та OEM-номером. При кліку на точку відбувається перехід безпосередньо на картку відповідної запчастини в каталозі.

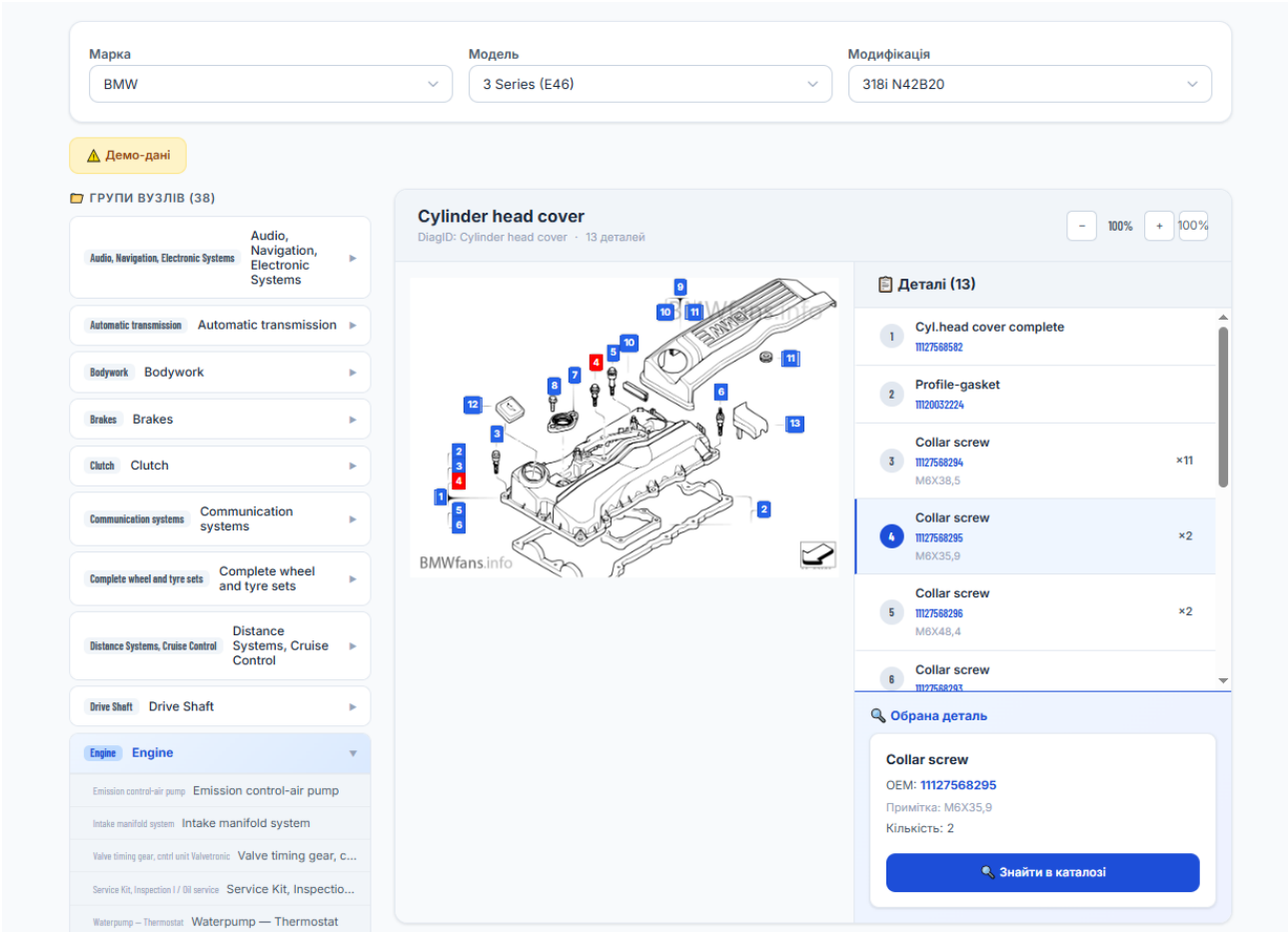


Рис. 3.17 – Інтерактивна діаграма вузла автомобіля з hotspot-точками

Адміністративна панель побудована на окремому вкладеному маршруті /admin та доступна виключно користувачам з роллю Admin. Вона має власну бічну навігаційну панель із розділами: Дашборд, Запчастини, Бренди,

Категорії, Автомобілі, Діаграми, Регламенти ТО, Партнерські локації та Користувачі. Головним елементом адміністративного розділу є аналітичний дашборд (рис. 3.18), що відображає агреговану статистику на основі даних таблиці SearchHistory: топ-5 найпопулярніших категорій (за кількістю переглядів), топ-5 брендів та топ-5 автомобілів у вигляді стовпчастих діаграм. Це дозволяє адміністратору оперативно оцінювати попит та приймати рішення щодо пріоритетів поповнення каталогу.

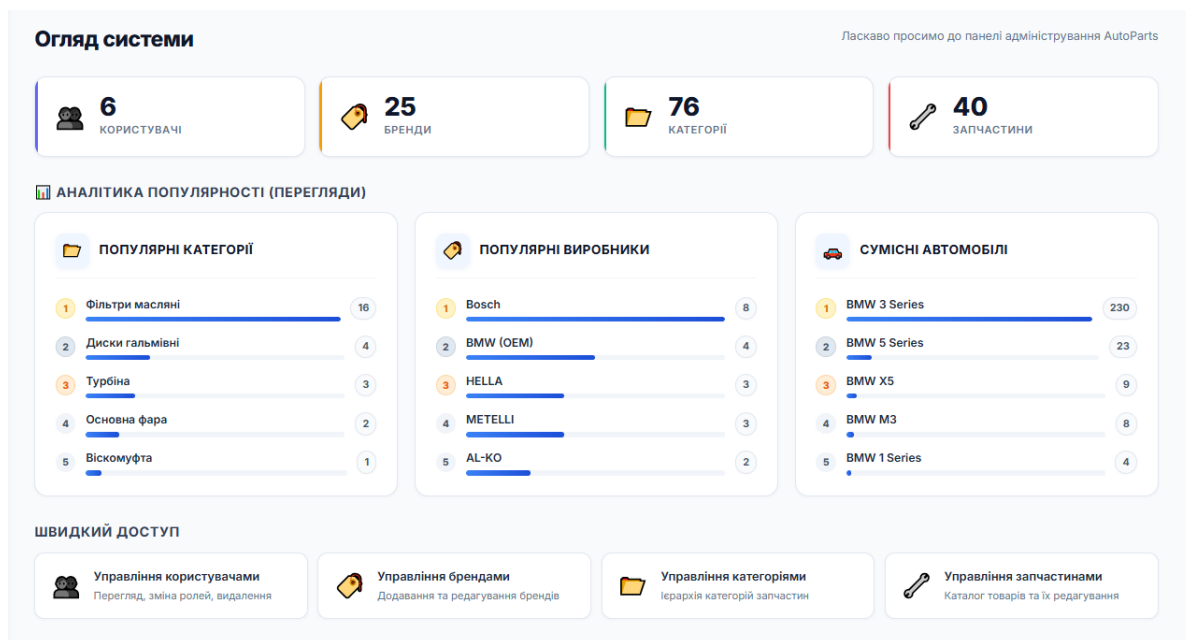


Рис. 3.18 – Адміністративна панель: аналітичний дашборд

Фундаментом для коректної роботи каталогу товарів є модуль управління ієрархією «Категорії» (рис. 3.19). Цей розділ адміністративної панелі забезпечує візуалізацію та зручне редагування багаторівневого дерева класифікації автозапчастин. Технічна реалізація цього функціоналу базується на патерні рекурсивних зв'язків типу «батьківська-дочірня категорія» (Parent-Child) у базі даних, що дозволяє системі підтримувати структуру з абсолютно будь-якою глибиною вкладеності без необхідності зміни схеми БД.

В інтерфейсі адміністратора ця ієрархія представлена у вигляді розкривного дерева. Для зручної навігації серед великого масиву записів передбачено рядок швидкого пошуку за назвою, а також глобальні елементи керування «Розгорнути всі» та «Згорнути всі». Кожен рядок (вузол) дерева відображає назву категорії, її унікальний ідентифікатор та містить панель

швидких дій. Адміністратор може відредагувати назву чи іконку, видалити категорію, або ж натиснути спеціальну кнопку «+ Під» (додати підкатегорію). В останньому випадку система автоматично відкриває форму створення нової категорії, одразу призначаючи обраний вузол як батьківський, що суттєво пришвидшує процес наповнення та структурування глобального каталогу платформи.

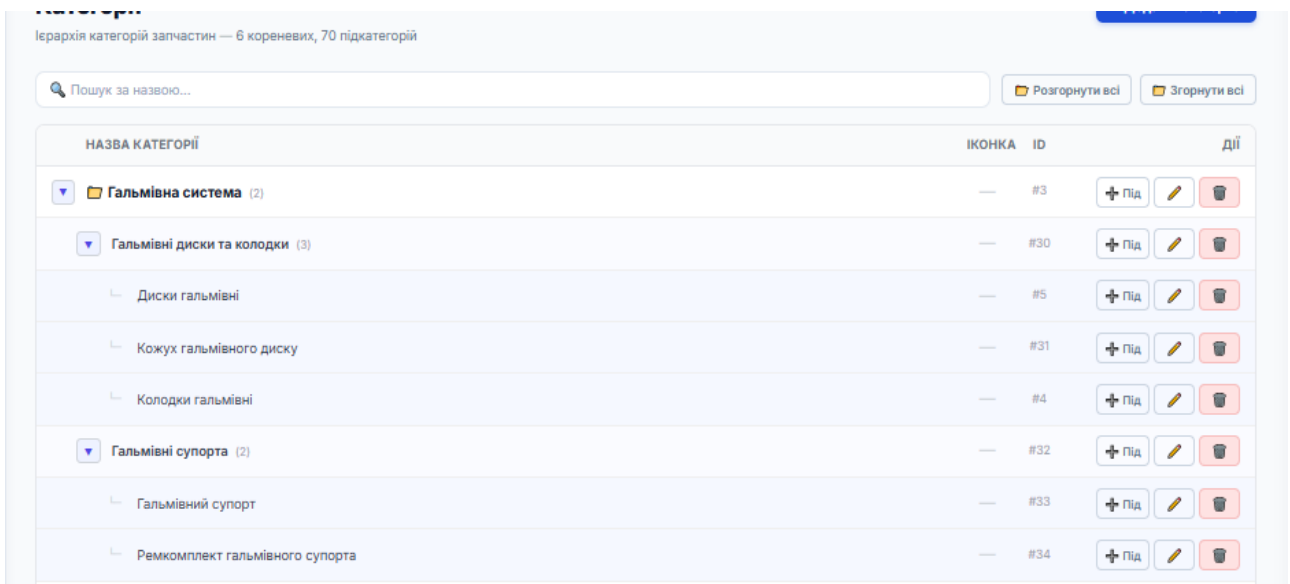


Рис. 3.19 – Адміністративна панель: управління ієрархічним деревом категорій запчастин

Сторінка управління запчастинами /admin/parts містить таблицю із переліком всіх позицій каталогу та кнопку «Додати запчастину». Форма додавання оснащена функцією автозаповнення (рис. 3.20): при введенні артикулу (SKU) у спеціальне поле та натисканні кнопки «Автозаповнити» система виконує запит до BMW Parts Catalog API та автоматично заповнює поля назви, бренду, OEM-номерів, технічних характеристик, переліку сумісних автомобілів та завантажує URL зображення. Адміністратор має можливість перевірити та відредагувати отримані дані перед збереженням. Для кожної існуючої позиції доступні кнопки редагування та видалення. Також для кожної запчастини доступна кнопка «Синхронізувати аналоги», що запускає автоматичний пошук аналогів через BMW Parts Catalog API та OEM-перетин у локальній базі даних.

Рис. 3.20 – Адміністративна панель: форма додавання запчастини з автозаповненням

Окремим потужним інструментом адміністративної панелі є модуль управління інтерактивними діаграмами вузлів автомобіля (рис. 3.21). Цей функціонал дозволяє не лише створювати ієрархічні структури схем з нуля, але й автоматизовано імпортувати (парсити) готові діаграми та переліки деталей із зовнішніх довідників, зокрема з каталогу BMWFans. Інтерфейс модуля логічно поділений на зону каскадного вибору конфігурації транспортного засобу (марка, модель, модифікація), навігаційне дерево доступних вузлів та робочу область налаштування поточної схеми.

В основній робочій області адміністратор має можливість гнучко керувати метаданими: передбачено поля для ручного редагування назви групи, підгрупи та безпосередньо назви самої діаграми перед збереженням у базу даних. Ключовою особливістю технічної реалізації цього модуля є інтерактивний механізм позиціювання деталей. Під завантаженим зображенням схеми формується таблиця з переліком прив'язаних запчастин із зазначенням позиції, опису, OEM-номера та кількості. Адміністратор може ініціювати режим встановлення мітки для будь-якої позиції з таблиці та клікнути на відповідне місце безпосередньо на зображенні діаграми. Система автоматично розраховує

відносні відсоткові координати кліку та зберігає їх, що в подальшому забезпечує абсолютно точне відображення інтерактивних точок (hotspots) для кінцевих користувачів клієнтського застосунку незалежно від розміру їхнього екрана.

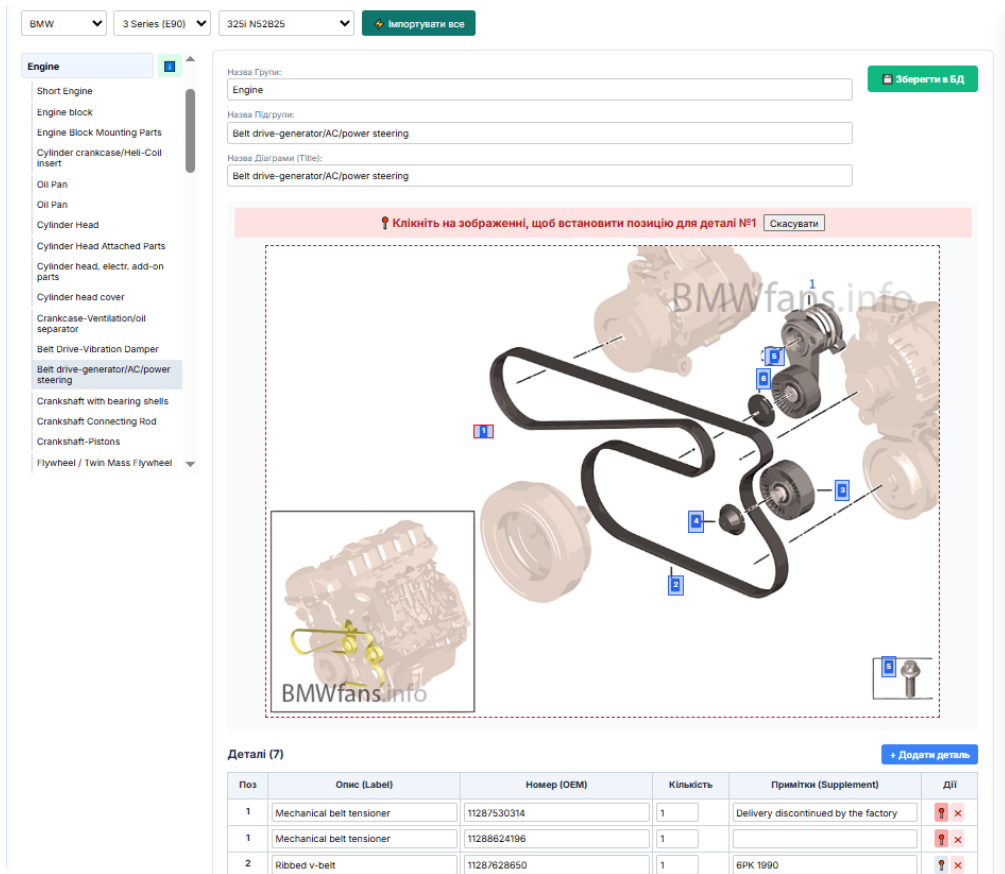


Рис. 3.21 – Адміністративна панель: інтерфейс імпорту та інтерактивного налаштування діаграм вузлів

Розділ «Магазини-партнери та Автосервіси» (рис. 3.22) призначений для пошуку та імпорту нових точок продажу запчастин і СТО. У верхній частині доступний перемикач робочих режимів: пошук нових локацій (через інтеграцію з Overpass) та управління вже збереженими записами [28].

Під ним розташована панель фільтрів, де можна вказати місто, радіус пошуку, тип об'єкта або вручну поставити мітку на мапі. Основна робоча область візуально розділена навпіл: зліва відображається інтерактивна мапа (Leaflet) із маркерами знайдених локацій, а праворуч — панель результатів. Вона містить лічильник знайдених об'єктів та список карток із детальною інформацією: назвою, контактними даними (адресою, телефоном) і бейджами типу чи статусу («Магазин», «Сервіс», «Імпортовано»).

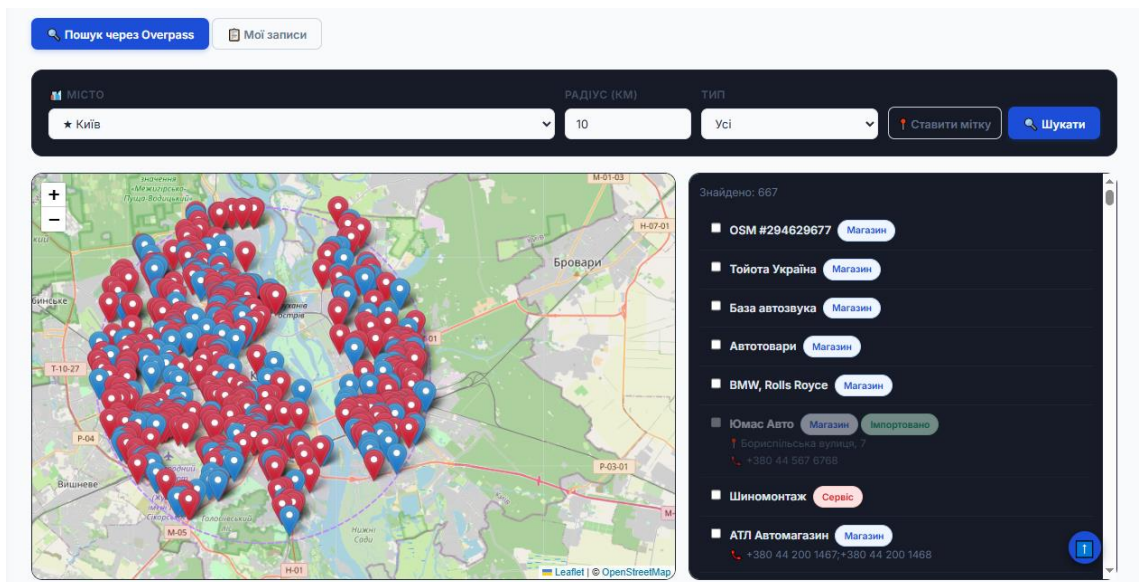


Рис. 3.22 – Адміністративна панель: блок «Партнерів»

Ще одним важливим і технічно складним компонентом адміністративної панелі є блок управління технічним обслуговуванням (ТО), який логічно розділений на два модулі: «Категорії запчастин для ТО» та «Регламенти ТО».

Модуль категорій (рис. 3.23) дозволяє адміністратору формувати структурований перелік типів деталей, що підлягають плановій заміні. Інтерфейс надає інструменти для прив'язки цих категорій до існуючих розділів глобального каталогу, додавання описової інформації, вибору іконок та налаштування порядку їх відображення (сортування) у клієнтському інтерфейсі.

Ці категорії відображаються у співвідношенні зі списком "Запчастини для ТО" в Гарні користувача. Кожна категорія може бути прив'язана до конкретної марки/моделі/модифікації або діяти для всіх авто. Більш специфічні категорії мають пріоритет над загальними.

Фільтр: **Всі (10)** **Загальні (5)** **По марці (2)** **По моделі (2)** **По модифікації (1)** **Всі марки**

Пошук за назвою, описом, маркою... 10 записів

№	ІКОНКА	НАЗВА	РІВЕНЬ ЗАСТОСУВАННЯ	ОПИС	КАТЕГОРІЯ КАТАЛОГУ	ПОРЯДОК	ДІЯ
4		Масло	Марка: Audi	—	Гальмівна система	0	
11		Фільтри масляні	Марка: BMW	загальні BMW	Фільтри масляні	0	
18		Фільтри масляні	Модель: BMW 3 Series F30	Для F30	Фільтри масляні	0	
15		Фільтри	Загальне (всі авто)	Заміна всіх фільтрів раз у 7000км	Фільтри	1	
14		Олива	Загальне (всі авто)	—	Не прив'язано	2	
1		Фільтр масляні	Загальне (всі авто)	Заміна кожні 6500 км, разом з маслом	Фільтри масляні	2	
12		Фільтр оливи	Модель: BMW 3 Series E46	Для e46	Фільтри масляні	2	
13		Фільтр оливи	Модифікація: BMW 3 Series E46 (318i)	для 318i	Фільтри масляні	2	
2		Гальмівні колодки	Загальне (всі авто)	—	Колодки гальмівні	5	
3		Диски гальмівні	Загальне (всі авто)	—	Диски гальмівні	5	

Рис. 3.23 – Адміністративна панель: управління категоріями запчастин для ТО

Модуль регламентів (рис. 3.24) призначений для безпосереднього налаштування інтервалів обслуговування. Для кожного компонента адміністратор встановлює нормативний пробіг у кілометрах, часовий інтервал у місяцях та формує супровідну текстову рекомендацію, яка згодом буде автоматично надсилатися користувачеві в електронному листі.

КОМПОНЕНТ	РІВЕНЬ ЗАСТОСУВАННЯ	ІНТЕРВАЛ (КМ)	ІНТЕРВАЛ (МІС)	РЕКОМЕНДАЦІЯ	ДІЇ
Фільтр масляний	Модифікація: BMW 3 Series (318i)	6 500 км	12 міс.	разом з маслом за допуском 5w3...	 
Ремінь ГРМ	Модифікація: BMW 3 Series (318i)	60 000 км	-	Разом з ремнем рекомендується...	 
ГРМ	Модифікація: BMW 3 Series (330i)	250 000 км	-	Не заплікати...	 
Фільтра	Модель: BMW 3 Series	6 666 км	6 міс.	6666...	 
Фільтра	Марка: BMW	7 000 км	14 міс.	12435...	 
Олива моторна (масло)	Загальне (всі авто)	6 500 км	18 міс.	-	 

Рис. 3.24 – Адміністративна панель: налаштування регламентів та інтервалів ТО

Ключовою архітектурною особливістю обох модулів є імплементація алгоритму каскадної рівнів застосування. Адміністратор має можливість створювати правила різного ступеня точності: від базових (для всіх транспортних засобів без винятку) до високоспецифічних (прив'язаних до конкретної марки, моделі або ж окремої модифікації автомобіля). В інтерфейсі ці рівні наочно виділяються відповідними кольоровими мітками-бейджами із зазначенням прив'язки.

Під час роботи клієнтського застосунку, коли система аналізує автомобіль з «Віртуального гаража», застосовується динамічний механізм пошуку найрелевантнішого правила (fallback-логіка). Алгоритм спочатку шукає регламент, створений спеціально для поточної модифікації авто; у разі його відсутності запит піднімається на рівень вище і перевіряє наявність правила для моделі, далі — для марки, і лише якщо жодних специфічних налаштувань не знайдено, система застосовує загальне правило. Такий гнучкий підхід дозволяє оптимізувати базу даних: адміністратору достатньо задати базові інтервали обслуговування як стандарти за замовчуванням, і точково створювати специфічні правила лише для тих автомобілів чи двигунів, які мають унікальні сервісні вимоги, що унеможлиблює інформаційну надмірність.

3.3. Тестування роботи програмного продукту

Для забезпечення надійності системи реалізовано багаторівневу стратегію тестування, що включає модульне тестування ізольованої бізнес-логіки, інтеграційне тестування REST API та ручне функціональне тестування. Автоматизовані тести інтегровано в конвеєр CI/CD у межах завдання Backend Build, Test & SonarCloud, що гарантує безперервний контроль якості при кожному оновленні коду. Модульні тести побудовано на базі фреймворку xUnit та бібліотеки Moq [1]. Для гарантії незалежності тестів від реального середовища використовується in-memory провайдер бази даних із генерацією унікальних імен контексту, а також mock-об'єкти для імітації зовнішніх залежностей, зокрема поштового сервісу EmailService.

Перевірка бізнес-логіки охоплює декілька ключових модулів. Зокрема, у модулі помічника технічного обслуговування здійснюється валідація алгоритму автоматичних сповіщень. Позитивний сценарій підтверджує генерацію електронного листа шляхом перевірки виклику методу відправки повідомлення із коректними параметрами, коли залишок до планового сервісу становить 500 кілометрів, що є меншим за встановлений поріг у 2000 кілометрів. Негативний сценарій підтверджує блокування відправки повідомлення, якщо запас пробігу значно перевищує вказаний поріг і становить, наприклад, 5000 кілометрів.

Іншим важливим аспектом є перевірка алгоритму оцінювання запчастин у відповідному контролері. Тест ініціалізує три типи запчастин з різними ціновими та якісними характеристиками для перевірки коректності автоматичного розрахунку показника ValueScore та присвоєння мітки найкращої пропозиції. Наприклад, для оригінальної преміум-запчастини вартістю 1000 гривень цей показник розраховується з урахуванням спеціального коефіцієнта і дорівнює 3.9. Водночас для стандартного неоригінального аналога вартістю 500 гривень показник сягає 4.0, що визначає його як оптимальний вибір за співвідношенням ціни та якості. Для найдешевшого бюджетного аналога вартістю 400 гривень показник становить лише 2.5 через нижчий рівень якості. Такий підхід у

тестуванні підтверджує, що алгоритм безпомилково розраховує дані, порівнює їх та виділяє запчастину з найвищим рейтингом (табл. 3.5).

Таблиця 3.5

Розрахунок ValueScore для юніт-тесту алгоритму підбору аналогів

Запчастина	QualityTier	IsOriginal	Ціна (грн)	Формула розрахунку	ValueScore
Original Part	1 (Преміум)	Так	1000	$(4-1) \times 1.3 / 1000 \times 1000$	3.9
Standard Analog	2 (Стандарт)	Ні	500	$(4-2) / 500 \times 1000$	4.0
Budget Analog	3 (Бюджет)	Ні	400	$(4-3) / 400 \times 1000$	2.5

Як видно з результатів розрахунку, найвищий показник отримує стандартний аналог, і відповідний тест перевіряє, що саме ця запчастина у відповіді програмного інтерфейсу позначається спеціальним прапорцем найкращого вибору, підтверджуючи тим самим коректність роботи алгоритму оцінювання.

Для інтеграційного тестування застосовується підхід із використанням фабрики вебдодатків, що дозволяє запустити повноцінний HTTP-сервер в оперативній пам'яті без необхідності його фізичного розгортання та надсилати до нього реальні мережеві запити. Перший інтеграційний тест звертається до базового маршруту отримання списку запчастин, підтверджуючи повернення статусу успішного виконання та правильного формату даних, що свідчить про коректне налаштування програмного стека та безпомилкову реєстрацію всіх залежностей системи. Другий інтеграційний тест виконує запит до завідомо неіснуючого ресурсу та очікує повернення статусу про відсутність даних замість внутрішньої помилки сервера, що підтверджує надійність механізму обробки виключень у бізнес-логіці.

Успішне проходження всіх описаних етапів перевірки, включаючи модульні та інтеграційні тести, автоматично фіксується у хмарному середовищі, що наочно демонструється статусом успішного виконання всіх завдань конвеєра безперервної інтеграції на платформі GitHub (рис. 3.25) [10].

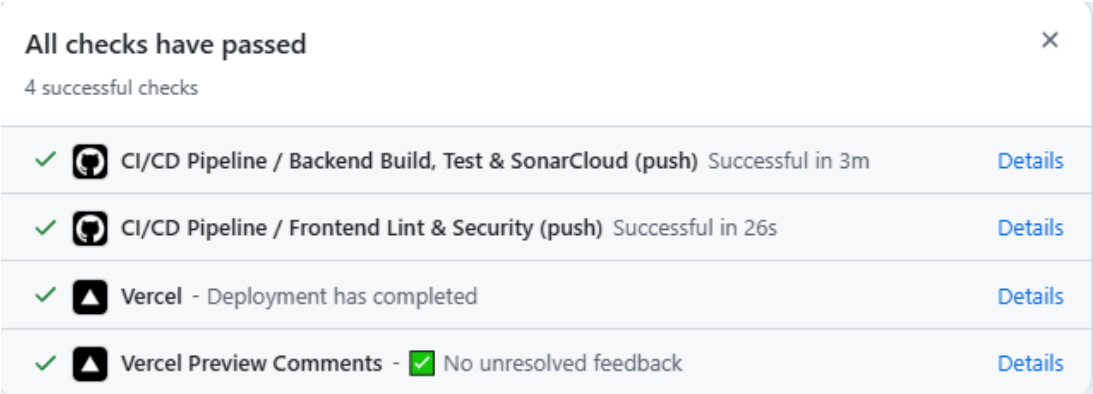


Рис. 3.25 – Результати успішного виконання автоматизованих перевірок CI/CD на GitHub

Для забезпечення високого рівня якості вихідного коду та дотримання стандартів безпеки, у межах налаштованого конвеєра безперервної інтеграції використовується хмарна платформа статичного аналізу SonarCloud. Під час виконання завдання збірки серверної частини система автоматично сканує кодову базу та формує детальний звіт щодо надійності, безпеки та зручності підтримки (рис. 3.26).

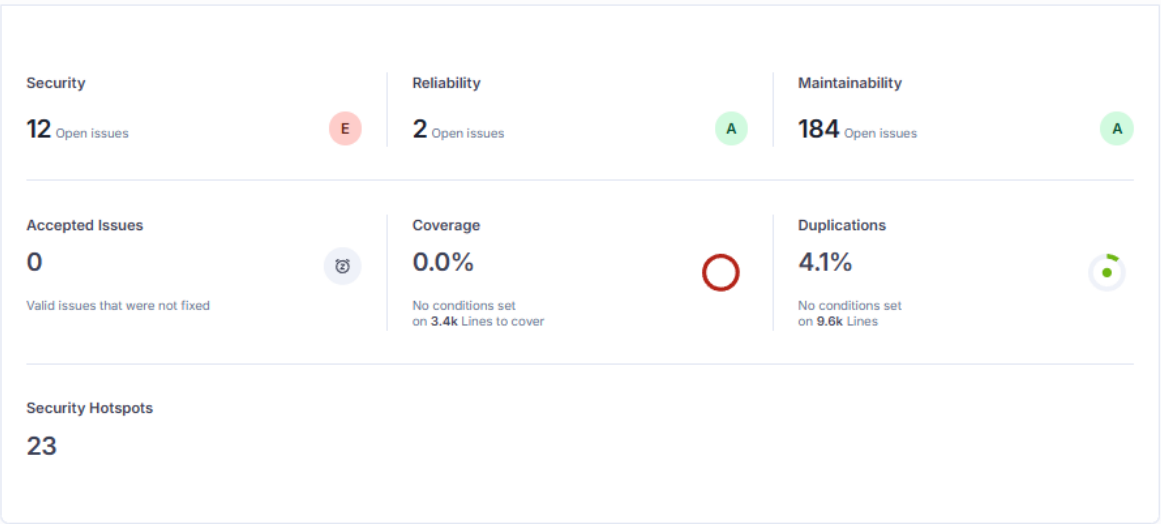


Рис. 3.26 – Результати статичного аналізу якості коду на платформі SonarCloud

Як видно з наведених результатів аналізу, платформа високо оцінила показники надійності (Reliability) та підтримуваності (Maintainability) коду, присвоївши їм найвищий рейтинг «А», а також зафіксувала цілком допустимий відсоток дублювання коду на рівні 4.1%. Водночас інструментарій виявив ряд потенційних вразливостей (Security Hotspots) та відкритих питань, які потребують додаткового ревію. Також звіт вказує на відсутність метрик покриття (Coverage). Використання такого автоматизованого інструментарію дозволяє оперативно виявляти технічний борг, відстежувати метрики якості у динаміці та проактивно усувати вразливості ще на етапі злиття коду, що є стандартом сучасної методології розробки програмного забезпечення.

Окрім автоматизованих тестів, для кожного ключового функціонального модуля проводилось ручне тестування у браузері Google Chrome. Результати ручного функціонального тестування зведено у таблицю 3.6.

Таблиця 3.6

Результати ручного функціонального тестування системи AutoParts

№	Модуль	Сценарій тестування	Очікуваний результат	Фактичний результат	Статус
1	Авторизація	Реєстрація нового користувача (email + пароль)	Профіль створено, JWT-токен отримано	Відповідає очікуваному	Пройдено
2	Авторизація	Вхід через Google OAuth 2.0	Акаунт знайдено/створено, вхід виконано	Відповідає очікуваному	Пройдено
3	Авторизація	Відновлення паролю через email	6-значний код надіслано, пароль змінено	Відповідає очікуваному	Пройдено
4	Пошук	Пошук за SKU через Elasticsearch	Результати відображено за < 2 секунди	Відповідає очікуваному	Пройдено
5	Пошук	Автодоповнення при введенні 2+ символів	Підказки з назвою, SKU та ціною	Відповідає очікуваному	Пройдено
6	Каталог	Фільтрація за spec-keys у категорії	Агрегація доступних значень характеристик	Відповідає очікуваному	Пройдено
7	Каталог	Фільтрація за сумісністю (VehicleSelector)	Сумісні деталі позначено міткою	Відповідає очікуваному	Пройдено

8	Картка деталі	Перегляд аналогів із ValueScore	Найкращий варіант виділено міткою	Відповідає очікуваному	Пройдено
9	Гараж	Додавання авто через VIN-декодування	Марку та модель зіставлено з БД автоматично	Відповідає очікуваному	Пройдено
10	Гараж	Оновлення пробігу → email-рекомендації ТО	Лист з рекомендаціями надіслано на email	Відповідає очікуваному	Пройдено
11	Діаграми	Hotspot-точка на SVG-схемі вузла	Перехід на картку запчастини при кліку	Відповідає очікуваному	Пройдено
12	Локації	Пошук магазинів через Overpass API/OSM	Результати на інтерактивній карті	Відповідає очікуваному	Пройдено
13	Адмін-панель	Автозаповнення картки через BMWAPI	Назва, OEM-номери, характеристики заповнено	Відповідає очікуваному	Пройдено
14	Безпека	Прямий GET-запит до /admin/users без JWT	Повернено 401 Unauthorized	Відповідає очікуваному	Пройдено
15	Безпека	Запит до /admin/users з JWT полі Customer	Повернено 403 Forbidden	Відповідає очікуваному	Пройдено
16	Адаптивність	Viewport < 768px (мобільний браузер)	Меню-drawer відображається коректно	Відповідає очікуваному	Пройдено

Усі шістнадцять сценаріїв ручного тестування демонструють поведінку системи, що повністю відповідає функціональним та нефункціональним вимогам, задекларованим у підрозділі 2.1.

Висновки до третього розділу

У третьому розділі описано практичну реалізацію системи AutoParts. Визначено вимоги до середовища та процедуру інсталяції: розгортання Elasticsearch у Docker, налаштування конфігурації, ініціалізацію бази даних PostgreSQL 15 через міграції EF Core та підключення клієнтської частини.

Організовано процес безперервної інтеграції на базі GitHub Actions зі стратегією Feature Branch. Головною особливістю конвеєра є автоматичне

розгортання повноцінної тестової бази даних для виконання інтеграційних перевірок у хмарному середовищі.

Описано архітектуру інтерфейсу з усіма маршрутами та ключові функціональні модулі: двоканальний пошук, каталог із динамічними фільтрами, «Віртуальний гараж» із декодуванням VIN, помічник технічного обслуговування, інтерактивні діаграми вузлів та мапу партнерів.

Надійність програмного забезпечення підтверджено трирівневим тестуванням. Модульні тести довели точність роботи алгоритмів аналізу ТО та розрахунку рейтингу ValueScore. Інтеграційні тести верифікували коректність роботи API, а шістнадцять сценаріїв ручного тестування остаточно підтвердили відповідність системи всім вимогам, надійність захисту доступу та правильність адаптивного відображення на мобільних пристроях.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проаналізовано предметну область підбору та продажу автозапчастин, що дозволило визначити ключові вимоги до майбутньої системи. Сформовано функціональні та нефункціональні вимоги, зокрема щодо точності перевірки сумісності деталей, швидкодії повнотекстового пошуку, безпеки та зручності користувацького інтерфейсу.

У першому розділі проведено огляд існуючих інформаційно-пошукових систем, виявлено їхні недоліки та обґрунтовано вибір клієнт-серверної архітектури. Обрано оптимальний технологічний стек: C# та ASP.NET Core для побудови REST API, PostgreSQL як основне сховище, Elasticsearch для швидкого пошуку, а також React для клієнтської частини.

Другий розділ присвячено проєктуванню системи: визначено основні сутності, побудовано доменну модель та розроблено схему реляційної бази даних із 25 таблиць. Спроектовано ключові функціональні модулі: двоканальний пошук, інтелектуальний помічник технічного обслуговування, «Віртуальний гараж» із декодуванням VIN-коду, інтерактивні схеми вузлів та мапу партнерських СТО. Описано алгоритми бізнес-логіки, зокрема оптимізований розрахунок рейтингу аналогів запчастин (ValueScore).

У третьому розділі розглянуто процес реалізації, розгортання та тестування платформи. Описано підготовку інфраструктури за допомогою Docker та міграцій EF Core. Налаштовано конвеєр безперервної інтеграції (CI/CD) у GitHub Actions для автоматизації збірки й аналізу коду. Підтверджено надійність системи шляхом комплексного тривіневого тестування: модульного, інтеграційного та ручного функціонального.

У результаті створено комплексну інформаційно-пошукову систему AutoParts із серверною, клієнтською та інфраструктурною частинами. Платформа ефективно вирішує задачі точного підбору деталей та моніторингу технічного стану авто. Застосовані архітектурні рішення забезпечують масштабованість, безпеку та готовність системи до практичного використання і подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Freeman A. Pro ASP.NET Core 8. 11th ed. Berkeley: Apress, 2024. 1200 p.
2. Albahari J. C# 12 in a Nutshell. Sebastopol: O'Reilly Media, 2023. 1090 p.
3. Troelsen A., Japikse P. Pro C# 11 with .NET 7. 12th ed. Berkeley: Apress, 2023. 1324 p.
4. Ковальчук В.О., Савченко О.В. Сучасні архітектурні патерни проєктування вебзастосунків на базі Clean Architecture. Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки. 2023. Т. 34, № 2. С. 45–51. DOI: 10.32838/2663-5941/2023.2-1/08.
5. Мельникова Н. І., Ткаченко О. М. Оптимізація запитів у реляційних базах даних на прикладі PostgreSQL 15. Інформаційні технології та комп'ютерна інженерія. 2023. Т. 48, № 1. С. 55–62. DOI: 10.31649/1999-9941-2023-48-1-55-62.
6. Мельник А.О., Коваль В. І. Оцінка продуктивності систем повнотекстового пошуку на базі Elasticsearch. Сучасні інформаційні системи. 2021. Т. 5, № 1. С. 12–19. DOI: 10.20998/2522-9052.2021.1.02.
7. Палеха О. О., Сидоренко В. М. Адаптація патерну управління станом для розробки інтерфейсів користувача за допомогою бібліотеки React 18. Сучасні проблеми інформатики та обчислювальної техніки. 2022. Т. 18, № 3. С. 112–119. DOI: 10.32620/reks.2022.3.10.
8. Kravchenko I. V., Shevchenko Y.O. Architecture of Modern E-commerce Platforms: Performance and Scalability. Science and Innovation. 2022. Т. 18, № 4. С. 23–31. DOI: 10.15407/scin18.04.023.
9. Ткаченко В.В., Романенко А.О. Методи захисту вебзастосунків від поширених вразливостей з використанням JWT та OAuth 2.0. Захист інформації. 2023. Т. 25, № 2. С. 88–95. DOI: 10.18372/2410-7840.25.14690.
10. Міненко В.В., Карпов І.В. Конвеєри безперервної інтеграції: впровадження CI/CD на базі GitHub Actions та Docker. Інженерія програмного забезпечення. 2023. № 4. С. 34–41. DOI: 10.32782/2522-9052.2023.4.05.

11. Коваленко С.М., Сидорчук О.П. Використання хмарних технологій CDN для зберігання та розповсюдження медіаконтенту у вебзастосунках. Вісник Національного університету «Львівська політехніка». Серія: Інформаційні системи та мережі. 2022. № 9. С. 89–96. DOI: 10.23939/sisn2022.09.089.
12. Левченко О.В., Петрик В.І. Оптимізація REST API за допомогою технологій кешування та асинхронної обробки в середовищі .NET. Комп'ютерні науки та кібербезпека. 2024. Т. 8, № 2. С. 45–52. DOI: 10.32782/2522-9052.2024.2.06.
13. Грицюк Ю. І., Рак Т.Є. Проєктування клієнт-серверних застосунків на базі React та Vite: навч. Посіб. Львів: Вид-во ЛДУ БЖД, 2024. 215 с.
14. Мартиненко О.В. Сучасні нереляційні та реляційні бази даних: навч. Посіб. Київ: КПІ ім. Ігоря Сікорського, 2022. 245 с.
15. Бойко В.М., Шевчук О.С. Особливості застосування технології Entity Framework Core для складних реляційних структур. Проблеми програмування. 2022. № 1. С. 55–63. DOI: 10.15407/pp2022.01.055.
16. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 3rd ed. Sebastopol: O'Reilly Media, 2024. 350 p.
17. Гавриленко О.В., Демченко В.І. Імплементація алгоритмів повнотекстового пошуку з використанням нечіткої логіки. Штучний інтелект. 2023. № 2. С. 41–48. DOI: 10.15407/jai2023.02.041.
18. Коваленко А.О., Козаченко Ю.В. Застосування патерну Dependency Injection у сучасних вебфреймворках. Системи управління, навігації та зв'язку. 2022. Вип. 4. С. 78–84. DOI: 10.26906/SUNZ.2022.4.078.
19. Литвин В.В., Гошко С.В. Архітектурні рішення для забезпечення високої доступності вебсервісів на базі Docker. Радіоелектроніка, інформатика, управління. 2023. № 1. С. 112–118. DOI: 10.15588/1607-3274-2023-1-12.
20. Данильченко А.М., Кравчук О.М. Аналіз продуктивності інструментів збірки фронтенд-застосунків. Вчені записки ТНУ імені В. І. Вернадського.

Серія: Технічні науки. 2024. Т. 35, № 1. С. 60–66. DOI: 10.32838/2663-5941/2024.1-1/10.

21. Павленко В. Д., Савченко Т. В. Моделювання бізнес-процесів електронної комерції з використанням UML та інтерактивних діаграм. Вісник Хмельницького національного університету. Технічні науки. 2021. № 5. С. 134–140. DOI: 10.31891/2307-5732-2021-299-5-134-140.
22. Олійник Ю.О., Мороз І.В. Використання геоінформаційних систем у вебзастосунках на прикладі OpenStreetMap API. Інформаційні технології: наука, техніка, технологія, освіта, здоров'я. 2022. Ч. 1. С. 150–156. URL: <http://science.kpi.kharkov.ua/wp-content/uploads/2022/05/MicroCAD-2022-Part-1.pdf> (дата звернення: 10.05.2026).
23. Entity Framework Core 8 Documentation. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/ef/core/> (дата звернення: 10.05.2026).
24. React 18 – Reference Overview. React Official Documentation. URL: <https://react.dev/reference/react> (дата звернення: 12.05.2026).
25. Vite: Next Generation Frontend Tooling. Vite Official Guide. URL: <https://vitejs.dev/guide/> (дата звернення: 15.05.2026).
26. Docker Documentation: Docker Compose V2. Docker Inc. URL: <https://docs.docker.com/compose/> (дата звернення: 18.05.2026).
27. Product Information Catalog and Vehicle API. National Highway Traffic Safety Administration (NHTSA). URL: <https://vpic.nhtsa.dot.gov/api/> (дата звернення: 20.05.2026).
28. OpenStreetMap Overpass API Documentation. OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/wiki/Overpass_API (дата звернення: 21.05.2026).
29. Cloudinary Programmable Media. Cloudinary Official Documentation. URL: <https://cloudinary.com/documentation> (дата звернення: 22.05.2026).
30. Elasticsearch Guide [8.12]. Elastic. URL: <https://www.elastic.co/guide/en/elasticsearch/reference/8.12/index.html> (дата звернення: 23.05.2026).

ДОДАТКИ

					ПЗ.КР.Б – 121 – 26 - ПЗ	Арк.
						78

Лістинг Part.cs (доменна сутність):

```

namespace AutoPartsSystem.Domain.Entities
{
    public class Part
    {
        public int Id { get; set; }
        public string SKU { get; set; } = string.Empty;           // Артикул виробника
        public string Name { get; set; } = string.Empty;
        public string? Description { get; set; }
        public decimal? Price { get; set; }
        public string? ImageUrl { get; set; }

        public double? WeightKg { get; set; }
        public double? LengthMm { get; set; }
        public double? WidthMm { get; set; }
        public double? HeightMm { get; set; }

        // Масиви номерів
        public List<string> OemNumbers { get; set; } = new(); // Оригінальні номери
        виробників авто
        public List<string> EanCodes { get; set; } = new();    // Штрих-коди

        public string? SpecificationsJson { get; set; }

        public int BrandId { get; set; }
        public Brand Brand { get; set; } = null!;

        public int CategoryId { get; set; }
        public Category Category { get; set; } = null!;

        public ICollection<PartCompatibility> Compatibilities { get; set; } = new
        List<PartCompatibility>();
        public ICollection<PartAnalog> Analogs { get; set; } = new List<PartAnalog>();
        public ICollection<Review> Reviews { get; set; } = new List<Review>();
    }
}

```

Лістинг ElasticPartSearchService.cs:

```

using AutoPartsSystem.Application.DTOs;
using AutoPartsSystem.Application.Interfaces;
using Elastic.Clients.Elasticsearch;
using Microsoft.Extensions.Logging;

namespace AutoPartsSystem.Infrastructure.Search
{
    public class ElasticPartSearchService : IPartSearchService
    {
        private readonly ElasticsearchClient _client;
        private readonly ILogger<ElasticPartSearchService> _logger;
        private const string IndexName = "autoparts";

        public ElasticPartSearchService(ElasticsearchClient client,
        ILogger<ElasticPartSearchService> logger)
        {
            _client = client;
            _logger = logger;
        }

        public async Task IndexPartAsync(PartSearchDocument document)
        {
            var response = await _client.IndexAsync(document, idx =>
            idx.Index(IndexName));
            if (!response.IsSuccess())

```

```

        {
            _logger.LogError("Помилка індексації: {DebugInformation}",
response.DebugInformation);
        }
    }

    public async Task<IEnumerable<PartSearchDocument>> SearchAsync(string query)
    {
        var response = await _client.SearchAsync<PartSearchDocument>(s => s
            .Index(IndexName)
            .Query(q => q
                .MultiMatch(m => m
                    .Query(query)
                    .Fields(new[]
                        {
                            "sku^3.0",           // Артикул найважливіший
                            "name^2.0",           // Назва на другому місці
                            "brandName",
                            "categoryName",
                            "crossNumbers",       // Аналоги (ОЕМ)
                            "specificationsText"
                        })
                    .Fuzziness(new Fuzziness("AUTO")) // слова з одруківками
                )
            );
        if (!response.IsSuccess())
        {
            _logger.LogError("Помилка пошуку: {DebugInformation}",
response.DebugInformation);
            return new List<PartSearchDocument>();
        }

        return response.Documents;
    }

    public async Task DeletePartAsync(int partId)
    {
        var response = await _client.DeleteAsync<PartSearchDocument>(partId, d =>
d.Index(IndexName));
        if (!response.IsSuccess() && response.ElasticsearchServerError?.Status !=
404)
        {
            _logger.LogError("Помилка при видаленні з Elasticsearch:
{DebugInformation}", response.DebugInformation);
        }
    }
}

```

Лістинг AutoPartsDbContext.cs – Fluent API: 25 таблиць, унікальні індекси, ієрархії, JSONB:

```

using AutoPartsSystem.Domain.Entities;
using Microsoft.EntityFrameworkCore;

namespace AutoPartsSystem.Infrastructure.Data
{
    public class AutoPartsDbContext : DbContext
    {

```

```

        public AutoPartsDbContext(DbContextOptions<AutoPartsDbContext> options) :
base(options) { }

        public DbSet<Make> Makes { get; set; }
        public DbSet<Model> Models { get; set; }
        public DbSet<Modification> Modifications { get; set; }
        public DbSet<Engine> Engines { get; set; }
        public DbSet<Transmission> Transmissions { get; set; }
        public DbSet<BodyType> BodyTypes { get; set; }
        public DbSet<Domain.Entities.DriveType> DriveTypes { get; set; }

        public DbSet<Part> Parts { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Brand> Brands { get; set; }
        public DbSet<PartCompatibility> PartCompatibilities { get; set; }
        public DbSet<PartAnalog> PartAnalogues { get; set; }
        public DbSet<Role> Roles { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<UserVehicle> UserVehicles { get; set; }
        public DbSet<Review> Reviews { get; set; }
        public DbSet<SearchHistory> SearchHistories { get; set; }
        public DbSet<PasswordResetToken> PasswordResetTokens { get; set; }

        public DbSet<MaintenanceRule> MaintenanceRules { get; set; }
        public DbSet<MaintenanceCategory> MaintenanceCategories { get; set; }

        // Діаграми вузлів авто
        public DbSet<VehicleGroup> VehicleGroups { get; set; }
        public DbSet<VehicleSubGroup> VehicleSubGroups { get; set; }
        public DbSet<DiagramImage> DiagramImages { get; set; }
        public DbSet<DiagramHotspot> DiagramHotspots { get; set; }

        // Магазины-партнери та автосервіси
        public DbSet<PartnerLocation> PartnerLocations { get; set; }
        public DbSet<PartnerLocationCountry> PartnerLocationCountries { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            // Налаштування JSONB для специфікацій запчастин
            modelBuilder.Entity<Part>()
                .Property(p => p.SpecificationsJson)
                .HasColumnType("jsonb");

            // Налаштування ієрархії категорій

```

```

modelBuilder.Entity<Category>()
    .HasOne(c => c.ParentCategory)
    .WithMany(c => c.SubCategories)
    .HasForeignKey(c => c.ParentCategoryId)
    .OnDelete(DeleteBehavior.Restrict);

// Унікальний індекс для артикулу (SKU) в межах одного бренду
modelBuilder.Entity<Part>()
    .HasIndex(p => new { p.SKU, p.BrandId })
    .IsUnique();

// Налаштування зв'язку для аналогів
modelBuilder.Entity<PartAnalog>()
    .HasOne(pa => pa.Part)
    .WithMany(p => p.Analogs)
    .HasForeignKey(pa => pa.PartId)
    .OnDelete(DeleteBehavior.Restrict);

modelBuilder.Entity<PartAnalog>()
    .HasOne(pa => pa.AnalogPart)
    .WithMany()
    .HasForeignKey(pa => pa.AnalogPartId)
    .OnDelete(DeleteBehavior.Restrict);

// Унікальний Email для юзера
modelBuilder.Entity<User>()
    .HasIndex(u => u.Email)
    .IsUnique();

// Налаштування гаража
modelBuilder.Entity<UserVehicle>()
    .HasOne(uv => uv.User)
    .WithMany(u => u.GarageVehicles)
    .HasForeignKey(uv => uv.UserId)
    .OnDelete(DeleteBehavior.Cascade);

// Налаштування ієрархії Авто
modelBuilder.Entity<Make>()
    .HasMany(m => m.Models)
    .WithOne(mo => mo.Make)
    .HasForeignKey(mo => mo.MakeId)
    .OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<Model>()
    .HasMany(m => m.Modifications)
    .WithOne(mo => mo.Model)

```

```

        .HasForeignKey(mo => mo.ModelId)
        .OnDelete(DeleteBehavior.Cascade);

// Модифікації з довідниками
modelBuilder.Entity<Modification>(entity =>
{
    entity.HasOne(m => m.Engine)
        .WithMany(e => e.Modifications)
        .HasForeignKey(m => m.EngineId);

    entity.HasOne(m => m.Transmission)
        .WithMany(t => t.Modifications)
        .HasForeignKey(m => m.TransmissionId);

    entity.HasOne(m => m.BodyType)
        .WithMany(b => b.Modifications)
        .HasForeignKey(m => m.BodyTypeId);

    entity.HasOne(m => m.DriveType)
        .WithMany(d => d.Modifications)
        .HasForeignKey(m => m.DriveTypeId);
});

// Гараж Користувача
modelBuilder.Entity<UserVehicle>()
    .HasOne(uv => uv.Modification)
    .WithMany(m => m.UserVehicles)
    .HasForeignKey(uv => uv.ModificationId);

// сумісності запчастин
modelBuilder.Entity<PartCompatibility>()
    .HasOne(pc => pc.Modification)
    .WithMany(m => m.Compatibilities)
    .HasForeignKey(pc => pc.ModificationId);

// Відгуки
modelBuilder.Entity<Review>()
    .HasOne(r => r.Part)
    .WithMany(p => p.Reviews)
    .HasForeignKey(r => r.PartId)
    .OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<Review>()
    .HasOne(r => r.User)
    .WithMany(u => u.Reviews)
    .HasForeignKey(r => r.UserId)

```

```

        .onDelete>DeleteBehavior.Cascade);

modelBuilder.Entity<Review>()
    .HasOne(r => r.ParentReview)
    .WithMany(r => r.Replies)
    .HasForeignKey(r => r.ParentReviewId)
    .onDelete>DeleteBehavior.Restrict);

// Діаграми вузлів авто
modelBuilder.Entity<VehicleGroup>()
    .HasMany(g => g.SubGroups)
    .WithOne(sg => sg.VehicleGroup)
    .HasForeignKey(sg => sg.VehicleGroupId)
    .onDelete>DeleteBehavior.Cascade);

modelBuilder.Entity<VehicleSubGroup>()
    .HasMany(sg => sg.DiagramImages)
    .WithOne(di => di.VehicleSubGroup)
    .HasForeignKey(di => di.VehicleSubGroupId)
    .onDelete>DeleteBehavior.Cascade);

modelBuilder.Entity<DiagramImage>()
    .HasMany(di => di.Hotspots)
    .WithOne(h => h.DiagramImage)
    .HasForeignKey(h => h.DiagramImageId)
    .onDelete>DeleteBehavior.Cascade);

modelBuilder.Entity<DiagramImage>()
    .HasOne(di => di.Modification)
    .WithMany()
    .HasForeignKey(di => di.ModificationId)
    .onDelete>DeleteBehavior.SetNull);

modelBuilder.Entity<DiagramHotspot>()
    .HasOne(h => h.Part)
    .WithMany()
    .HasForeignKey(h => h.PartId)
    .onDelete>DeleteBehavior.SetNull);

// Категорії ТО
modelBuilder.Entity<MaintenanceCategory>(entity =>
{
    entity.HasOne(mc => mc.Category)
        .WithMany()
        .HasForeignKey(mc => mc.CategoryId)
        .onDelete>DeleteBehavior.SetNull);
}

```

```

        entity.HasOne(mc => mc.Make)
            .WithMany()
            .HasForeignKey(mc => mc.MakeId)
            .OnDelete(DeleteBehavior.Cascade);

        entity.HasOne(mc => mc.Model)
            .WithMany()
            .HasForeignKey(mc => mc.ModelId)
            .OnDelete(DeleteBehavior.Cascade);

        entity.HasOne(mc => mc.Modification)
            .WithMany()
            .HasForeignKey(mc => mc.ModificationId)
            .OnDelete(DeleteBehavior.Cascade);

        // Унікальність: назва + рівень застосування
        entity.HasIndex(mc => new { mc.Name, mc.MakeId, mc.ModelId,
mc.ModificationId })
            .IsUnique();
    });

    // Магазины-партнери та автосервіси
    modelBuilder.Entity<PartnerLocation>(entity =>
    {
        entity.HasMany(pl => pl.Countries)
            .WithOne(c => c.PartnerLocation)
            .HasForeignKey(c => c.PartnerLocationId)
            .OnDelete(DeleteBehavior.Cascade);

        // Унікальний індекс на OsmNodeId для запобігання дублікатів з Overpass
        entity.HasIndex(pl => pl.OsmNodeId)
            .IsUnique()
            .HasFilter("\"OsmNodeId\" IS NOT NULL");
    });

    // Tokens for password reset
    modelBuilder.Entity<PasswordResetToken>(entity =>
    {
        entity.HasOne(t => t.User)
            .WithMany()
            .HasForeignKey(t => t.UserId)
            .OnDelete(DeleteBehavior.Cascade);

        entity.HasIndex(t => new { t.UserId, t.Token });
    });
}

```

```

    }
}

```

Лістинг PartController.cs – алгоритм ValueScore:

```

[HttpGet("{id}/analogs-with-value")]
[AllowAnonymous]
public async Task<IActionResult> GetAnalogsWithBestValue(int id)
{
    var part = await _context.Parts
        .Include(p => p.Brand)
        .FirstOrDefaultAsync(p => p.Id == id);

    if (part == null) return NotFound("Запчастину не знайдено.");

    var directAnalog = await _context.PartAnalog
        .Where(pa => pa.PartId == id)
        .Include(pa => pa.AnalogPart).ThenInclude(p => p.Brand)
        .Select(pa => new
        {
            pa.AnalogPart.Id,
            pa.AnalogPart.SKU,
            pa.AnalogPart.Name,
            Brand = pa.AnalogPart.Brand.Name,
            BrandId = pa.AnalogPart.Brand.Id,
            QualityTier = pa.AnalogPart.Brand.QualityTier,
            IsBrandOriginal = pa.AnalogPart.Brand.IsOriginal,
            pa.AnalogPart.Price,
            pa.AnalogPart.ImageUrl,
            pa.MatchQuality
        })
        .ToListAsync();

    var reverseAnalog = await _context.PartAnalog
        .Where(pa => pa.AnalogPartId == id)
        .Include(pa => pa.Part).ThenInclude(p => p.Brand)
        .Select(pa => new
        {
            pa.Part.Id,
            pa.Part.SKU,
            pa.Part.Name,
            Brand = pa.Part.Brand.Name,
            BrandId = pa.Part.Brand.Id,
            QualityTier = pa.Part.Brand.QualityTier,
            IsBrandOriginal = pa.Part.Brand.IsOriginal,
            pa.Part.Price,
            pa.Part.ImageUrl,
            pa.MatchQuality
        })
        .ToListAsync();

    var candidates = directAnalog
        .Concat(reverseAnalog)
        .DistinctBy(a => a.Id)
        .ToListAsync();

    var allCandidates = candidates
        .Select(a => new
        {
            a.Id,
            a.SKU,
            a.Name,
            a.Brand,
            a.BrandId,
            a.QualityTier,
            a.IsBrandOriginal,
            a.Price,

```

```

        a.ImageUrl,
        a.MatchQuality,
        IsOriginal = false
    })
    .Append(new
    {
        part.Id,
        part.SKU,
        part.Name,
        Brand = part.Brand.Name,
        BrandId = part.Brand.Id,
        QualityTier = part.Brand.QualityTier,
        IsBrandOriginal = part.Brand.IsOriginal,
        part.Price,
        part.ImageUrl,
        MatchQuality = "Original",
        IsOriginal = true
    })
    .ToList();

// ValueScore: (4 - QualityTier) / Price
// Tier 1 (премiум) → множник 3, Tier 2 (стандарт) → множник 2, Tier 3
(бюджет) → множник 1
// Чим нижча ціна при вищій якості → тим вищий score

var validPrices = allCandidates.Where(c => c.Price > 0).Select(c =>
(double)c.Price).ToList();
double globalAvgPrice = validPrices.Any() ? validPrices.Average() : 1000.0;

var tierPrices = allCandidates.Where(c => c.Price > 0)
    .GroupBy(c => c.QualityTier)
    .ToDictionary(g => g.Key, g => g.Average(x => (double)x.Price));

int? bestValueId = null;
double maxScore = -1;

var scoredCandidates = allCandidates.Select(c =>
{
    double effectivePrice = c.Price > 0 ? (double)c.Price :
(tierPrices.ContainsKey(c.QualityTier) ? tierPrices[c.QualityTier] : globalAvgPrice);
    double qualityScore = 4.0 - c.QualityTier;
    if (c.IsBrandOriginal) qualityScore *= 1.3;

    double score = (qualityScore / effectivePrice) * 1000;

    return new
    {
        Candidate = c,
        EffectivePrice = effectivePrice,
        Score = score
    };
}).ToList();

foreach (var sc in scoredCandidates)
{
    if (sc.Candidate.Price > 0 && sc.Score > maxScore)
    {
        maxScore = sc.Score;
        bestValueId = sc.Candidate.Id;
    }
}

var result = scoredCandidates.Select(sc => new
{
    sc.Candidate.Id,
    sc.Candidate.SKU,
    sc.Candidate.Name,
    sc.Candidate.Brand,
    sc.Candidate.BrandId,

```

```

        sc.Candidate.QualityTier,
        QualityTierLabel = sc.Candidate.QualityTier == 1 ? "Преміум" :
sc.Candidate.QualityTier == 2 ? "Стандарт" : "Бюджет",
        sc.Candidate.IsBrandOriginal,
        sc.Candidate.Price,
        sc.Candidate.ImageUrl,
        sc.Candidate.MatchQuality,
        sc.Candidate.IsOriginal,
        IsBestValue = sc.Candidate.Id == bestValueId,
        ValueScore = Math.Round(sc.Score, 2)
    })
    .OrderByDescending(c => c.IsBestValue)
    .ThenByDescending(c => c.ValueScore)
    .ToList();

    return Ok(result);
}

```

Лістинг MaintenanceAssistantService.cs – Каскадний вибір регламентів ТО,

email-сповіщення:

```

using AutoPartsSystem.Application.Interfaces;
using AutoPartsSystem.Infrastructure.Data;
using Microsoft.EntityFrameworkCore;
using System.Text;

namespace AutoPartsSystem.Infrastructure.Services
{
    public class MaintenanceAssistantService : IMaintenanceAssistantService
    {
        private readonly AutoPartsDbContext _context;
        private readonly IEmailService _emailService;

        public MaintenanceAssistantService(AutoPartsDbContext context, IEmailService
emailService)
        {
            _context = context;
            _emailService = emailService;
        }

        public async Task AnalyzeAndNotifyAsync(int userVehicleId)
        {
            var garageVehicle = await _context.UserVehicles
                .Include(uv => uv.User)
                .Include(uv => uv.Modification)
                .ThenInclude(m => m.Model)
                .ThenInclude(model => model.Make)
                .FirstOrDefaultAsync(uv => uv.Id == userVehicleId);

            if (garageVehicle == null || !garageVehicle.CurrentMileage.HasValue)
return;

            int currentMileage = garageVehicle.CurrentMileage.Value;

            var allRules = await _context.MaintenanceRules
                .Where(r => r.ModificationId == garageVehicle.ModificationId ||
                    r.ModelId == garageVehicle.Modification.ModelId ||
                    r.MakeId == garageVehicle.Modification.Model.MakeId ||
                    (r.ModificationId == null && r.ModelId == null && r.MakeId
== null))
                .ToListAsync();

            if (!allRules.Any()) return;

            var specificRules = allRules

```

```

        .GroupBy(r => r.ComponentName)
        .Select(g => g.OrderByDescending(r => r.ModificationId.HasValue)
            .ThenByDescending(r => r.ModelId.HasValue)
            .ThenByDescending(r => r.MakeId.HasValue)
            .First())
        .ToList();

var recommendations = new List<string>();

// Математика ТО: що треба міняти найближчим часом
foreach (var rule in specificRules)
{
    if (!rule.IntervalKm.HasValue) continue;

    int intervalKm = rule.IntervalKm.Value;
    int prevServiceKm = (currentMileage / intervalKm) * intervalKm;
    int nextServiceKm = prevServiceKm + intervalKm;

    int kmSinceLastService = currentMileage - prevServiceKm;
    int kmToNextService = nextServiceKm - currentMileage;

    if (kmToNextService <= 2000 || kmSinceLastService <= 1000)
    {
        int targetServiceKm = kmSinceLastService <= 1000 ? prevServiceKm
: nextServiceKm;
        if (targetServiceKm == 0) continue;

        string monthInfo = rule.IntervalMonths.HasValue ? $" або
{rule.IntervalMonths.Value} міс." : "";
        string status = kmSinceLastService <= 1000 && kmSinceLastService
> 0
            ? $"прострочено на {kmSinceLastService} км"
            : $"залишилось {targetServiceKm - currentMileage} км";

        if (targetServiceKm == currentMileage) status = "саме час для ТО";

        recommendations.Add($"<li><b>{rule.ComponentName}</b> – регламент
на {targetServiceKm} км{monthInfo} ({status}). <i>{rule.Recommendation}</i></li>");
    }
}

if (!recommendations.Any()) return;

string carFullName = $"{garageVehicle.Modification.Model.Make.Name}
{garageVehicle.Modification.Model.Name}";

var sb = new StringBuilder();
sb.AppendLine($"<h3>Привіт, {garageVehicle.User.FirstName}</h3>");

sb.AppendLine($"<p>Твій автомобіль <b>{garageVehicle.Alias ??
carFullName}</b> наближається до планового технічного обслуговування (поточний пробіг:
{currentMileage} км).</p>");
sb.AppendLine("<p>За регламентом виробника ми рекомендуємо звернути увагу
на наступні вузли:</p>");
sb.AppendLine("<ul>");
foreach (var rec in recommendations) sb.AppendLine(rec);
sb.AppendLine("</ul>");

string catalogUrl = $"https://auto-parts-system-
diploma.vercel.app/catalog?makeId={garageVehicle.Modification.Model.MakeId}&modelId={garage
Vehicle.Modification.ModelId}&modificationId={garageVehicle.ModificationId}";
sb.AppendLine($"<p>Ви можете підібрати необхідні запчастини для вашого
авто в нашому каталозі за цим посиланням:</p>");
sb.AppendLine($"<p><a href=\"{catalogUrl}\" style=\"display: inline-block;
padding: 10px 20px; background-color: #0056b3; color: white; text-decoration: none; border-
radius: 6px; font-weight: bold;>Перейти до каталогу</a></p>");

await _emailService.SendEmailAsync(
    garageVehicle.User.Email,
    $"Рекомендації ТО: {carFullName}",

```

```

        sb.ToString()
    };
}
}
}

```

Лістинг UsersController.cs – JWT (HMAC-256); Google OAuth 2.0

авторизація:

```

[HttpPost("login")]
public async Task<IActionResult> Login([FromBody] LoginDto dto)
{
    var user = await _context.Users.Include(u => u.Role).FirstOrDefaultAsync(u
=> u.Email == dto.Email);

    if (user == null || !BCrypt.Net.BCrypt.Verify(dto.Password,
user.PasswordHash))
        return Unauthorized("Невірний email або пароль.");

    var claims = new[]
    {
        new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
        new Claim(ClaimTypes.Email, user.Email),
        new Claim(ClaimTypes.Role, user.Role.Name),
        new Claim("FullName", $"{user.FirstName} {user.LastName}")
    };

    var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var token = new JwtSecurityToken(
        issuer: _config["Jwt:Issuer"],
        audience: _config["Jwt:Audience"],
        claims: claims,
        expires: DateTime.Now.AddDays(1),
        signingCredentials: creds
    );

    return Ok(new { Token = new JwtSecurityTokenHandler().WriteToken(token)
});
}

[HttpPost("google-login")]
public async Task<IActionResult> GoogleLogin([FromBody] GoogleLoginDto dto)
{
    try
    {
        var payload = await
GoogleJsonWebSignature.ValidateAsync(dto.Credential);

        var user = await _context.Users.Include(u =>
u.Role).FirstOrDefaultAsync(u => u.Email == payload.Email);

        if (user == null)
        {
            var customerRole = await _context.Roles.FirstOrDefaultAsync(r =>
r.Name == "Customer");
            user = new User
            {
                FirstName = payload.GivenName ?? "Google",
                LastName = payload.FamilyName ?? "User",
                Email = payload.Email,
                PasswordHash =
BCrypt.Net.BCrypt.HashPassword(Guid.NewGuid().ToString()),
                RoleId = customerRole!.Id
            };
        }
    }
}

```

```

        };
        _context.Users.Add(user);
        await _context.SaveChangesAsync();
        user.Role = customerRole;
    }

    var claims = new[]
    {
        new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
        new Claim(ClaimTypes.Email, user.Email),
        new Claim(ClaimTypes.Role, user.Role.Name),
        new Claim("FullName", $"{user.FirstName} {user.LastName}")
    };

    var key = new
    SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var token = new JwtSecurityToken(
        issuer: _config["Jwt:Issuer"],
        audience: _config["Jwt:Audience"],
        claims: claims,
        expires: DateTime.Now.AddDays(1),
        signingCredentials: creds
    );

    return Ok(new { Token = new JwtSecurityTokenHandler().WriteToken(token)
});
}
catch (Exception ex)
{
    return BadRequest($"Помилка авторизації через Google: {ex.Message}");
}
}

```

ТЕСТУВАННЯ:

Лістинг MaintenanceAssistantServiceTests.cs – xUnit, Moq, InMemory:

```

using AutoPartsSystem.Application.Interfaces;
using AutoPartsSystem.Domain.Entities;
using AutoPartsSystem.Infrastructure.Data;
using AutoPartsSystem.Infrastructure.Services;
using Microsoft.EntityFrameworkCore;
using Moq;
using Xunit;

namespace AutoPartsSystem.Tests.Services
{
    public class MaintenanceAssistantServiceTests
    {
        private AutoPartsDbContext GetDbContext()
        {
            var options = new DbContextOptionsBuilder<AutoPartsDbContext>()
                .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())
                .Options;

            return new AutoPartsDbContext(options);
        }

        [Fact]
        public async Task AnalyzeAndNotifyAsync_ShouldSendEmail_WhenServiceIsDue()
        {
            // Arrange
            var dbContext = GetDbContext();
            var mockEmailService = new Mock<IEmailService>();

```

```

};
        var user = new User { Id = 1, FirstName = "Test", Email = "test@example.com"

        var make = new Make { Id = 1, Name = "BMW" };
        var model = new Model { Id = 1, Name = "X5", MakeId = 1, Make = make };
        var modification = new Modification { Id = 1, Name = "3.0d", ModelId = 1,
Model = model };

        var vehicle = new UserVehicle
        {
            Id = 1,
            UserId = 1,
            User = user,
            ModificationId = 1,
            Modification = modification,
            CurrentMileage = 29500 // Due for 30000km service
        };

        var rule = new MaintenanceRule
        {
            Id = 1,
            ComponentName = "Масло",
            IntervalKm = 10000,
            ModelId = 1,
            Recommendation = "Заміна масла"
        };

        dbContext.Users.Add(user);
        dbContext.Makes.Add(make);
        dbContext.Models.Add(model);
        dbContext.Modifications.Add(modification);
        dbContext.UserVehicles.Add(vehicle);
        dbContext.MaintenanceRules.Add(rule);
        await dbContext.SaveChangesAsync();

        var service = new MaintenanceAssistantService(dbContext,
mockEmailService.Object);

        // Act
        await service.AnalyzeAndNotifyAsync(1);

        // Assert
        mockEmailService.Verify(es => es.SendEmailAsync(
            "test@example.com",
            "Рекомендації TO: BMW X5",
            It.Is<string>(s => s.Contains("Масло") && s.Contains("30000 км"))
        ), Times.Once);
    }

    [Fact]
    public async Task
AnalyzeAndNotifyAsync_ShouldNotSendEmail_WhenServiceIsNotDue()
    {
        // Arrange
        var dbContext = GetDbContext();
        var mockEmailService = new Mock<IEmailService>();

        var user = new User { Id = 1, FirstName = "Test", Email = "test@example.com"
};
        var make = new Make { Id = 1, Name = "BMW" };
        var model = new Model { Id = 1, Name = "X5", MakeId = 1, Make = make };
        var modification = new Modification { Id = 1, Name = "3.0d", ModelId = 1,
Model = model };

        var vehicle = new UserVehicle
        {
            Id = 1,
            UserId = 1,
            User = user,

```

```

        ModificationId = 1,
        Modification = modification,
        CurrentMileage = 15000 // Not due for 30000km service
    };

    var rule = new MaintenanceRule
    {
        Id = 1,
        ComponentName = "Масло",
        IntervalKm = 10000,
        ModelId = 1,
        Recommendation = "Заміна масла"
    };

    dbContext.Users.Add(user);
    dbContext.Makes.Add(make);
    dbContext.Models.Add(model);
    dbContext.Modifications.Add(modification);
    dbContext.UserVehicles.Add(vehicle);

    dbContext.MaintenanceRules.Add(rule);
    await dbContext.SaveChangesAsync();

    var service = new MaintenanceAssistantService(dbContext,
mockEmailService.Object);

    // Act
    await service.AnalyzeAndNotifyAsync(1);

    // Assert
    mockEmailService.Verify(es => es.SendEmailAsync(
        It.IsAny<string>(),
        It.IsAny<string>(),
        It.IsAny<string>()
    ), Times.Never);
}
}
}

```

ЛІСТИНГ PartsControllerTests.cs – Перевірка ValueScore:

```

using AutoPartsSystem.Api.Controllers;
using AutoPartsSystem.Application.Interfaces;
using AutoPartsSystem.Domain.Entities;
using AutoPartsSystem.Infrastructure.Data;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Moq;
using System.Text.Json;
using Xunit;

namespace AutoPartsSystem.Tests.Controllers
{
    public class PartsControllerTests
    {
        private AutoPartsDbContext GetDbContext()
        {
            var options = new DbContextOptionsBuilder<AutoPartsDbContext>()
                .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())
                .Options;

            return new AutoPartsDbContext(options);
        }

        [Fact]
        public async Task GetAnalogsWithBestValue_ShouldCalculateValueScoreCorrectly()
        {
            // Arrange
            var dbContext = GetDbContext();

```

```

        var mockSearchService = new Mock<IPartSearchService>();

        var premiumBrand = new Brand { Id = 1, Name = "Premium", QualityTier = 1,
IsOriginal = true };
        var standardBrand = new Brand { Id = 2, Name = "Standard", QualityTier =
2, IsOriginal = false };
        var budgetBrand = new Brand { Id = 3, Name = "Budget", QualityTier = 3,
IsOriginal = false };

        var originalPart = new Part { Id = 1, SKU = "ORIG1", Name = "Original
Part", Price = 1000m, BrandId = 1, Brand = premiumBrand };
        var standardAnalog = new Part { Id = 2, SKU = "STD1", Name = "Standard
Analog", Price = 500m, BrandId = 2, Brand = standardBrand };
        var budgetAnalog = new Part { Id = 3, SKU = "BUD1", Name = "Budget Analog",
Price = 400m, BrandId = 3, Brand = budgetBrand };

        dbContext.Brands.AddRange(premiumBrand, standardBrand, budgetBrand);
        dbContext.Parts.AddRange(originalPart, standardAnalog, budgetAnalog);

        dbContext.PartAnalog.Add(new PartAnalog { Id = 1, PartId = 1, AnalogPartId
= 2, MatchQuality = "Good" });
        dbContext.PartAnalog.Add(new PartAnalog { Id = 2, PartId = 1, AnalogPartId
= 3, MatchQuality = "Good" });

        await dbContext.SaveChangesAsync();

        var controller = new PartsController(dbContext, mockSearchService.Object);

        // Act
        var result = await controller.GetAnalogsWithBestValue(1);

        // Assert
        var okResult = Assert.IsType<OkObjectResult>(result);
        var valueStr = JsonSerializer.Serialize(okResult.Value);
        var elements = JsonSerializer.Deserialize<JsonElement[]>(valueStr);

        Assert.NotNull(elements);
        Assert.Equal(3, elements.Length);
        var bestValueItem = elements.FirstOrDefault(e =>
e.GetProperty("IsBestValue").GetBoolean() == true);
        Assert.Equal(2, bestValueItem.GetProperty("Id").GetInt32()); //
standardAnalog Id = 2
        Assert.Equal(4.0, bestValueItem.GetProperty("ValueScore").GetDouble());
    }
}

```

Лістинг PartsApiIntegrationTests.cs – WebApplicationFactory:

```

using AutoPartsSystem.API;
using Microsoft.AspNetCore.Mvc.Testing;
using Xunit;
using System.Net;

namespace AutoPartsSystem.Tests.Integration
{
    public class PartsApiIntegrationTests :
    IClassFixture<WebApplicationFactory<Program>>
    {
        private readonly WebApplicationFactory<Program> _factory;

        public PartsApiIntegrationTests(WebApplicationFactory<Program> factory)
        {
            _factory = factory;
        }

        [Fact]
        public async Task GetParts_ShouldReturnSuccessStatusCode()

```

```

        {
            var client = _factory.CreateClient();
            var response = await client.GetAsync("/api/parts");
            response.EnsureSuccessStatusCode(); // Status Code 200-299
            Assert.Equal("application/json; charset=utf-8",
response.Content.Headers.ContentType?.ToString());
        }

        [Fact]
        public async Task GetPartById_ShouldReturnNotFound_ForInvalidId()
        {
            var client = _factory.CreateClient();
            var response = await client.GetAsync("/api/parts/999999");
            Assert.Equal(HttpStatusCode.NotFound, response.StatusCode);
        }
    }
}

```

Лістинг CloudinaryImageStorageService.cs – Завантаження в CDN з

АВТООПТИМІЗАЦІЄЮ ЯКОСТІ:

```

using AutoPartsSystem.Application.Interfaces;
using CloudinaryDotNet;
using CloudinaryDotNet.Actions;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

namespace AutoPartsSystem.Infrastructure.Services
{
    public class CloudinaryImageStorageService : IImageStorageService
    {
        private readonly Cloudinary _cloudinary;
        private readonly ILogger<CloudinaryImageStorageService> _logger;

        public CloudinaryImageStorageService(IConfiguration config,
ILogger<CloudinaryImageStorageService> logger)
        {
            var account = new Account(
                config["CloudinarySettings:CloudName"],
                config["CloudinarySettings:ApiKey"],
                config["CloudinarySettings:ApiSecret"]
            );

            _cloudinary = new Cloudinary(account);
            _logger = logger;
            _cloudinary.Api.Secure = true;
        }

        public async Task<string> UploadImageAsync(IFormFile file)
        {
            if (file == null || file.Length == 0)
                throw new ArgumentException("Файл порожній");

            using var stream = file.OpenReadStream();

            var uploadParams = new ImageUploadParams
            {
                File = new FileDescription(file.FileName, stream),
                Folder = "autoparts_catalog",
                Transformation = new Transformation().Quality("auto").FetchFormat("auto")
            };

            var uploadResult = await _cloudinary.UploadAsync(uploadParams);
        }
    }
}

```

```

        if (uploadResult.Error != null)
        {
            throw new Exception($"Помилка Cloudinary:
{uploadResult.Error.Message}");
        }

        return uploadResult.SecureUrl.ToString();
    }

    public async Task<string?> UploadFromUrlAsync(string imageUrl, string folder
= "diagram_images")
    {
        if (string.IsNullOrEmpty(imageUrl)) return null;

        try
        {
            var secureUrl = imageUrl.StartsWith("http://")
                ? "https://" + imageUrl.Substring(7)
                : imageUrl;

            var uploadParams = new ImageUploadParams
            {
                File = new FileDescription(secureUrl),
                Folder = folder,
                Transformation = new
Transformation().Quality("auto").FetchFormat("auto"),
                UniqueFilename = true
            };

            var result = await _cloudinary.UploadAsync(uploadParams);

            if (result.Error != null)
            {
                _logger.LogError("[Cloudinary] Upload error for {ImageUrl}:
{ErrorMessage}", imageUrl, result.Error.Message);
                return imageUrl;
            }

            return result.SecureUrl?.ToString();
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "[Cloudinary] Exception uploading {ImageUrl}:
{Message}", imageUrl, ex.Message);
            return imageUrl;
        }
    }
}

```