

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-сайту для агрегації пропозицій з сайтів
продажу автомобілів з пробігом»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень. Використання
ідей, результатів і текстів інших авторів мають посилання
на відповідне джерело*

_____ Максим ОСАДЧИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

_____ Максим ОСАДЧИЙ

Керівник: _____ Світлана ШЕВЧЕНКО
канд.пед.наук, доц.

Рецензент: _____

Київ 2026

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Осадчому Максиму Вікторовичу

1. Тема кваліфікаційної роботи: «Розробка web-сайту для агрегації пропозицій з сайтів продажу автомобілів з пробігом»

керівник кваліфікаційної роботи канд.пед.наук, доц. Світлана ШЕВЧЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про централізований моніторинг ринку автомобілів з пробігом, особливості пошуку, фільтрації, порівняння та оцінювання автомобільних оголошень, підходи до агрегації даних з різних web-платформ, принципи побудови клієнт-серверних web-застосунків, методи реалізації авторизації, аналітичної обробки даних і розрахунку показника вигідності пропозиції AutoHub Deal Score, а також технічна документація щодо використання Java, Spring Boot, Spring Data JPA, Spring

Security, JWT, PostgreSQL, Liquibase, React, Vite, REST API, Docker Compose, JUnit та Mockito.

4. Зміст розрахунково-пояснювальної записки:

1. Огляд та аналіз предметної галузі моніторингу ринку автомобілів з пробігом, особливостей пошуку, порівняння та оцінювання автомобільних оголошень.
2. Аналіз існуючих web-сервісів для продажу та моніторингу вживаних автомобілів, визначення їх переваг, недоліків і функціональних обмежень.
3. Постановка задачі розробки та формування функціональних і нефункціональних вимог до web-платформи AutoHub.
4. Проектування архітектури web-платформи, структури бази даних, ролей користувачів, основних модулів системи та логіки AutoHub Deal Score.
5. Програмна реалізація серверної та клієнтської частин web-платформи AutoHub, авторизації, каталогу, фільтрації, обраного, порівняння, аналітики, профілю користувача та адміністративної панелі.
6. Тестування основних функцій web-платформи та оцінка відповідності реалізованого програмного забезпечення поставленим вимогам.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів web-сервісів для пошуку та моніторингу вживаних автомобілів.
2. Функціональні та нефункціональні вимоги до web-платформи AutoHub.
3. Діаграма варіантів використання web-платформи AutoHub.
4. Загальна архітектура web-платформи AutoHub.
5. Діаграма взаємодії технологій проєкту.
6. ER-діаграма бази даних AutoHub.
7. Діаграма класів основної бізнес-логіки системи.
8. Діаграма класів авторизації та користувачів.
9. Діаграми послідовності основних сценаріїв: авторизація, пошук автомобіля, додавання до обраного, розрахунок AutoHub Deal Score.
10. Екранні форми web-платформи.
11. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури за темою централізованого моніторингу ринку автомобілів з пробігом	23.02–28.02.2026	
2	Аналіз та дослідження існуючих web-сервісів для продажу та моніторингу автомобілів з пробігом	01.03–07.03.2026	
3	Аналіз предметної галузі ринку автомобілів з пробігом та визначення вимог до web-платформи AutoHub	09.03–15.03.2026	
4	Огляд засобів реалізації web-платформи AutoHub	16.03–22.03.2026	
5	Проектування web-платформи AutoHub, архітектури системи, структури бази даних та UML-діаграм	23.03–05.04.2026	
6	Програмна реалізація серверної та клієнтської частин web-платформи AutoHub	06.04–26.04.2026	
7	Реалізація каталогу, фільтрації, обраного, порівняння, аналітики та механізму AutoHub Deal Score	27.04–03.05.2026	
8	Тестування web-платформи AutoHub та оформлення результатів тестування	04.05–10.05.2026	
9	Оформлення роботи: вступ, висновки, реферат, список джерел та додатки	11.05–17.05.2026	
10	Розробка демонстраційних матеріалів та підготовка до попереднього захисту роботи	18.05–22.05.2026	

Здобувач вищої освіти.

(підпис)

Максим ОСАДЧИЙ

Керівник
кваліфікаційної роботи

(підпис)

Світлана ШЕВЧЕНКО

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 85 стор., 3 табл., 23 рис., 25 джерел.

Мета роботи – поліпшення процесу пошуку автомобілів з пробігом за допомогою створення єдиної платформи, яка об'єднує дані з сайтів продажу авто, забезпечуючи можливість фільтрації, порівняння та моніторингу цін.

Об'єкт дослідження – процес пошуку та аналізу пропозицій продажу автомобілів з пробігом в онлайн-середовищі.

Предмет дослідження – розробка web-сайту для агрегації оголошень, фільтрації даних, порівняння характеристик автомобілів з пробігом та представлення аналітики цін.

Короткий зміст роботи: Дане дослідження присвячене розробці web-сайту AutoHub для агрегації пропозицій з сайтів продажу автомобілів з пробігом. У роботі здійснено аналіз існуючих аналогів додатків, таких як AUTO.RIA, OLX Авто, mobile.de та AutoScout24. Було визначено їхні сильні та слабкі сторони, що дозволило сформулювати вимоги до створення нового застосунку та його проектування. Розроблено алгоритм роботи застосунку та програмно реалізовані ключові функціональні можливості, зокрема: збір і відображення оголошень з різних сайтів; пошук і розширена фільтрація за параметрами авто; порівняння кількох пропозицій; відображення історії перевірки та ключових атрибутів; аналітика цін і виявлення схожих оголошень автомобілів за VIN кодом; збереження обраних варіантів і сповіщення про нові пропозиції

Особливістю розробленої системи є механізм AutoHub Deal Score, який дозволяє оцінювати вигідність конкретного автомобільного оголошення за 100-бальною шкалою. Під час розрахунку враховуються співвідношення ціни автомобіля із середньою ринковою ціною, пробіг відносно року випуску, актуальність оголошення, наявність історії зміни ціни та популярність марки або моделі в системі. Завдяки цьому користувач отримує не лише список оголошень, а

й аналітичну підказку щодо доцільності подальшого розгляду конкретної пропозиції.

У роботі використано Java та Spring Boot для серверної частини, PostgreSQL та Liquibase для бази даних, React для браузерної частини, JWT для автентифікації користувачів та Docker для контейнеризації.

Проведене функціональне тестування системи показало, що web-платформа відповідає зазначеним вимогам.

Користуватися web-сайтом можуть покупці, які бажають порівняти й знайти оптимальні варіанти вживаних авто, а також продавці, автодилери та автопідбирачі для оперативного моніторингу цін та аналізу вторинного ринку.

КЛЮЧОВІ СЛОВА: AUTONUB, АВТОМОБІЛІ З ПРОБІГОМ, WEB-ПЛАТФОРМА, МОНІТОРИНГ РИНКУ, АГРЕГАЦІЯ ДАНИХ, ПОШУК АВТО, ФІЛЬТРАЦІЯ, ПОРІВНЯННЯ АВТОМОБІЛІВ, DEAL SCORE, SPRING BOOT, REACT, POSTGRESQL, LIQUIBASE, JWT, DOCKER, REST API.

ЗМІСТ

РЕФЕРАТ.....	7
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	11
ВСТУП.....	13
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ МОНІТОРИНГУ РИНКУ	
АВТОМОБІЛІВ З ПРОБІГОМ.....	16
1.1 Поняття та особливості ринку автомобілів з пробігом.....	16
1.2 Учасники ринку та їх інформаційні потреби.....	17
1.3 Проблеми пошуку та порівняння автомобілів.....	18
1.4 Необхідність централізованої платформи AutoHub.....	19
1.5 Концепція AutoHub Deal Score як засобу підтримки прийняття рішення	21
2 АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ДЛЯ ПОШУКУ ТА	
МОНІТОРИНГУ АВТОМОБІЛІВ З ПРОБІГОМ.....	23
2.1 Огляд сервісу AUTO.RIA.....	24
2.2 Огляд сервісу OLX Авто.....	25
2.3 Огляд сервісу mobile.de.....	26
2.4 Огляд сервісу AutoScout24.....	27
2.5 Порівняльний аналіз функціональних можливостей аналогів.....	29
2.6 Виявлені недоліки існуючих систем.....	29
3 ПОСТАНОВКА ЗАДАЧІ ТА ФОРМУВАННЯ ВИМОГ ДО WEB-	
ПЛАТФОРМИ AUTONUB.....	31
3.1 Постановка задачі розробки.....	32
3.2 Функціональні вимоги.....	34
3.4 Нефункціональні вимоги.....	36
3.5 Ролі користувачів та сценарії доступу.....	37
4 ПРОЄКТУВАННЯ WEB-ПЛАТФОРМИ AUTONUB.....	40
4.1 Загальна архітектура системи.....	40
4.2 Архітектура клієнтської та серверної частини.....	42
4.3 Проєктування бази даних.....	45

4.4	Проектування логіки AutoHub Deal Score.....	48
4.5	Алгоритм розрахунку оцінки вигідності автомобіля.....	50
4.6	Діаграми послідовності основних сценаріїв.....	52
5	ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ПЛАТФОРМИ AUTONUB.....	56
5.1	Реалізація серверної частини.....	56
5.2	Реалізація клієнтської частини.....	60
5.3	Реалізація авторизації та розмежування ролей.....	63
5.4	Реалізація каталогу, фільтрації та порівняння автомобілів.....	64
5.5	Реалізація профілю користувача.....	66
5.6	Реалізація адміністративної панелі.....	68
5.7	Реалізація AutoHub Deal Score.....	70
6	ТЕСТУВАННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ РОБОТИ СИСТЕМИ.....	73
6.1	План тестування.....	73
6.2	Функціональне тестування.....	75
6.3	Перевірка ролей доступу.....	78
6.4	Оцінка роботи механізму Deal Score.....	79
6.5	Аналіз результатів.....	81
	ВИСНОВКИ.....	83
	ПЕРЕЛІК ПОСИЛАНЬ.....	86
	ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ.....	88
	ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ.....	98

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – інтерфейс прикладного програмування, що використовується для взаємодії між програмними компонентами або зовнішніми сервісами.

REST API – архітектурний підхід до організації взаємодії між клієнтською та серверною частинами web-застосунку через HTTP-запити.

HTTP – протокол передавання даних у мережі Інтернет.

JSON – текстовий формат обміну даними, який використовується для передавання структурованої інформації між frontend- і backend-частинами системи.

DTO – об'єкт передавання даних, який використовується для обміну інформацією між шарами застосунку або між сервером і клієнтом.

ORM – технологія об'єктно-реляційного відображення, що дозволяє працювати з таблицями бази даних через об'єкти мови програмування.

JPA – специфікація Java для роботи з реляційними базами даних через об'єктну модель.

CRUD – базові операції з даними: створення, читання, оновлення та видалення.

SQL – мова структурованих запитів, що використовується для роботи з реляційними базами даних.

Frontend – клієнтська частина web-застосунку, з якою безпосередньо взаємодіє користувач.

Backend – серверна частина web-застосунку, яка виконує бізнес-логіку, обробляє запити та працює з базою даних.

Java – об'єктно-орієнтована мова програмування, що використовується для реалізації серверної частини платформи AutoHub.

Spring Boot – фреймворк екосистеми Spring, який спрощує створення, налаштування та запуск backend-застосунків.

Spring Security – модуль Spring, який використовується для реалізації автентифікації, авторизації та розмежування доступу.

Spring Data JPA – модуль Spring для роботи з реляційними базами даних через JPA-репозиторії.

JWT – формат токена, що використовується для захищеної передачі даних про автентифікованого користувача між клієнтом і сервером.

BCrypt – алгоритм хешування паролів, який використовується для безпечного зберігання облікових даних користувачів.

React – бібліотека JavaScript для створення компонентного користувацького інтерфейсу.

Vite – інструмент для швидкого запуску, розробки та збирання frontend-застосунків.

PostgreSQL – об'єктно-реляційна система керування базами даних, що використовується для зберігання даних платформи AutoHub.

Liquibase – інструмент для керування змінами структури бази даних за допомогою міграцій.

Docker – платформа контейнеризації, що використовується для запуску компонентів системи в ізольованому середовищі.

Docker Compose – інструмент для опису та запуску багатоконтейнерних застосунків.

Maven – система збірки Java-проектів і керування залежностями.

Git – система контролю версій, що використовується для відстеження змін у програмному коді.

Swagger/OpenAPI – засоби для документування, опису та тестування REST API.

AutoHub – web-платформа для централізованого моніторингу, пошуку, порівняння та аналітичного оцінювання пропозицій на ринку автомобілів з пробігом.

Deal Score – показник оцінювання вигідності автомобільного оголошення, який враховує ціну, пробіг, актуальність оголошення, історію зміни ціни та популярність моделі.

Dashboard – аналітична сторінка або панель, на якій відображаються ключові показники роботи системи.

SPA – односторінковий web-застосунок, у якому взаємодія з користувачем відбувається без повного перезавантаження сторінки.

ВСТУП

У сучасних умовах цифровізації процес купівлі та продажу автомобілів з пробігом значною мірою перемістився в онлайн-середовище. Користувачі все частіше шукають автомобілі через спеціалізовані web-платформи, маркетплейси та сервіси оголошень. Водночас ринок автомобілів з пробігом залишається розподіленим між різними джерелами, кожне з яких має власну структуру оголошень, набір фільтрів, спосіб відображення інформації та підхід до роботи з користувачами [1 – 6].

Актуальність теми кваліфікаційної роботи зумовлена необхідністю спрощення процесу пошуку, порівняння та оцінювання автомобільних оголошень. У реальних умовах користувачу часто доводиться відкривати декілька сайтів, вручну порівнювати ціни, пробіг, рік випуску, комплектацію, регіон продажу та актуальність оголошення. Такий підхід потребує значних часових витрат і не завжди дозволяє швидко визначити, яка пропозиція є найбільш вигідною [1 - 4].

Проблема розрізненості ринку автомобілів з пробігом полягає в тому, що оголошення розміщуються на різних платформах, а користувач не має єдиного середовища для їхнього централізованого перегляду та аналітичного порівняння. Навіть, якщо окремі сервіси надають зручний пошук і фільтрацію, вони зазвичай обмежені власною базою даних і не забезпечують повної картини ринку. Крім того, більшість існуючих рішень не надає прозорого механізму оцінювання вигідності конкретного автомобіля [1 – 4].

У зв'язку з цим актуальною є розробка web-платформи AutoHub, яка забезпечує централізований моніторинг ринку автомобілів з пробігом, зберігання оголошень в єдиній базі даних, пошук, фільтрацію, порівняння характеристик, роботу з обраними пропозиціями та перегляд аналітики. Такий підхід дозволяє не лише скоротити час пошуку автомобіля, а й підвищити якість прийняття рішення за рахунок структурованого представлення даних [1 – 6].

Особливо важливим елементом системи є механізм AutoHub Deal Score. Він дозволяє оцінювати вигідність конкретного автомобільного оголошення за 100-

бальною шкалою з урахуванням ціни відносно середньої ринкової вартості, пробігу, року випуску, актуальності оголошення, історії зміни ціни та популярності марки або моделі. Завдяки цьому користувач отримує не просто список автомобілів, а аналітичну підказку щодо доцільності подальшого розгляду конкретної пропозиції [1 – 4]. Цим підтверджується актуальність та важливість даного дослідження.

Мета роботи – поліпшення процесу пошуку автомобілів з пробігом за допомогою створення єдиної платформи, яка об'єднує дані з сайтів продажу авто, забезпечуючи можливість фільтрації, порівняння та моніторингу цін.

Об'єкт дослідження – процес пошуку та аналізу пропозицій продажу автомобілів з пробігом в онлайн-середовищі.

Предмет дослідження – розробка web-сайту для агрегації оголошень, фільтрації даних, порівняння характеристик автомобілів з пробігом та представлення аналітики цін.

Для досягнення поставленої мети необхідно вирішити такі *завдання*:

1. Провести аналіз предметної галузі моніторингу ринку автомобілів з пробігом та визначити основні проблеми пошуку, порівняння й оцінювання автомобільних оголошень.
2. Дослідити існуючі програмні рішення для продажу та моніторингу автомобілів з пробігом, порівняти їх функціональні можливості та визначити основні недоліки.
3. Сформулювати функціональні та нефункціональні вимоги до web-сайту для агрегації пропозицій з сайтів продажу автомобілів з пробігом.
4. Дослідити інструменти та технології для розробки web-сайту для агрегації пропозицій з сайтів продажу автомобілів з пробігом.
5. Розробити web-сайт для агрегації пропозицій з сайтів продажу автомобілів з пробігом на основі обраних технологій і з урахуванням зазначених вимог.

6. Провести тестування web-сайту для агрегації пропозицій з сайтів продажу автомобілів з пробігом на основі обраних технологій і з урахуванням зазначених вимог.

Методи дослідження. Для досягнення поставлених завдань у роботі використовуються наступні методи: системно-структурний аналіз, порівняльний метод, а також методи передачі, зберігання і обробки даних. Для розробки застосунку застосовані сучасні технології: Java та Spring Boot для серверної частини, PostgreSQL та Liquibase для бази даних, React для браузерної частини, JWT для автентифікації користувачів, та Docker для контейнеризації.

Практична значущість одержаних результатів полягає в тому, що розроблена web-платформа може бути використана як основа для подальшого розвитку системи моніторингу ринку автомобілів з пробігом. AutoHub дозволяє об'єднати в одному інтерфейсі каталог автомобілів, пошук, фільтрацію, перегляд деталей, обране, порівняння, аналітику та оцінювання вигідності пропозицій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ МОНІТОРИНГУ РИНКУ АВТОМОБІЛІВ З ПРОБІГОМ

1.1 Поняття та особливості ринку автомобілів з пробігом

Ринок автомобілів з пробігом є складною та динамічною предметною галуззю, у якій щоденно формується велика кількість пропозицій від приватних продавців, автосалонів, дилерів та спеціалізованих онлайн-платформ. На відміну від ринку нових автомобілів, де характеристики транспортного засобу є більш стандартизованими, у сфері продажу автомобілів з пробігом кожне оголошення має індивідуальні параметри: технічний стан, пробіг, рік випуску, комплектацію, регіон продажу, історію експлуатації, ціну та джерело походження інформації [1 - 6].

Особливістю цієї галузі є те, що інформація про автомобілі не зберігається в одному централізованому середовищі. Користувачі змушені самостійно переглядати різні сайти оголошень, порівнювати ціни, перевіряти актуальність пропозицій та аналізувати, чи відповідає конкретний автомобіль середньоринковим умовам. Через це процес вибору автомобіля часто стає тривалим, незручним і потребує від користувача значної кількості ручних дій.

До ключових параметрів, які впливають на вибір вживаного автомобіля, належать марка, модель, рік випуску, пробіг, ціна, тип кузова, тип пального, трансмісія, об'єм двигуна, регіон продажу та наявність додаткової інформації про транспортний засіб. Однак навіть за наявності цих характеристик користувач не завжди може швидко визначити, чи є конкретна пропозиція вигідною. Наприклад, автомобіль може мати нижчу ціну, але водночас більший пробіг або старішу дату публікації оголошення. Саме тому для ефективного аналізу ринку потрібні не лише інструменти пошуку, а й засоби порівняння та аналітичної оцінки пропозицій [1 - 4].

Ринок вживаних авто також характеризується високою мінливістю. Оголошення можуть швидко з'являтися, змінюватися або зникати з платформ. Ціни можуть оновлюватися залежно від попиту, стану автомобіля, сезону, курсу валют та активності продавця. Тому для користувача важливо бачити не тільки поточну ціну автомобіля, а й історію її зміни, що дає змогу краще оцінити поведінку пропозиції на ринку [1 - 4].

Таким чином, предметна галузь моніторингу ринку автомобілів з пробігом пов'язана з обробкою великого обсягу різнорідних даних, що надходять із різних джерел. Для її ефективної автоматизації необхідно створити систему, яка забезпечує централізоване збирання, зберігання, фільтрацію, порівняння та аналітичну обробку інформації про автомобільні оголошення [1 - 6].

1.2 Учасники ринку та їх інформаційні потреби

У процесі купівлі, продажу та аналізу вживаних автомобілів беруть участь різні групи користувачів, кожна з яких має власні цілі та інформаційні потреби. Для розробки web-платформи важливо визначити ці групи, оскільки саме від них залежить функціональність системи, структура інтерфейсу та логіка доступу до даних [1 - 6].

Першою групою є гості, тобто користувачі, які ще не пройшли авторизацію в системі. Вони зазвичай зацікавлені у швидкому перегляді каталогу автомобілів, ознайомленні з доступними пропозиціями, використанні базових фільтрів та перегляді детальної інформації про окремі оголошення. Для цієї категорії важливо забезпечити простий доступ до основного функціоналу без обов'язкової реєстрації [7].

Другою групою є зареєстровані користувачі. На відміну від гостей, вони потребують ширших можливостей для персональної роботи з оголошеннями. До таких можливостей належать додавання автомобілів до обраного, порівняння декількох пропозицій, перегляд персонального профілю, аналіз збережених авто та використання додаткових інструментів оцінювання. Зареєстрований

користувач розглядається як основний споживач функціональності системи, оскільки саме він активно взаємодіє з платформою під час вибору автомобіля [1 – 4].

Окрему роль у системі займає адміністратор. Його завдання полягає не у звичайному перегляді оголошень, а в контролі роботи платформи. Адміністратор повинен мати доступ до статистики системи, перегляду користувачів, керування оголошеннями, запуску оновлення даних та контролю актуальності інформації. Наявність адміністративної ролі дозволяє зробити систему більш керованою та наближеною до реального промислового web-застосунку [7].

До учасників предметної галузі також можна віднести покупців, продавців та аналітиків. Покупці зацікавлені у швидкому пошуку вигідних пропозицій і можливості порівняти автомобілі між собою. Продавці орієнтовані на розміщення привабливих оголошень і залучення потенційних клієнтів. Аналітики або досвідчені користувачі потребують узагальнених даних про ринок, середні ціни, популярні марки, зміну вартості автомобілів і динаміку пропозицій [1 - 4].

Інформаційні потреби цих груп можна узагальнити таким чином: користувачам необхідний зручний пошук, прозоре порівняння, доступ до повної інформації про автомобіль, можливість збереження цікавих варіантів та аналітична підтримка прийняття рішення. Саме ці потреби мають бути враховані під час проектування платформи AutoHub.

1.3 Проблеми пошуку та порівняння автомобілів

Однією з основних проблем ринку вживаних автомобілів є фрагментованість інформації. Оголошення розміщуються на різних платформах, кожна з яких має власну структуру даних, набір фільтрів, спосіб відображення характеристик і правила публікації. Через це користувач змушений працювати з кількома сайтами одночасно, відкривати багато вкладок, вручну порівнювати схожі автомобілі та самостійно зберігати цікаві варіанти [1 - 6].

Наступною проблемою є складність порівняння пропозицій. Навіть якщо два автомобілі мають однакову марку та модель, вони можуть суттєво відрізнятися за роком випуску, пробігом, комплектацією, регіоном продажу, типом двигуна або станом. У більшості існуючих сервісів користувач отримує лише список оголошень, але не має достатньої аналітичної підтримки для визначення, яке з них є більш вигідним [1 - 4].

Також суттєвою проблемою є відсутність єдиної логіки оцінювання. Користувач може бачити ціну автомобіля, але не завжди розуміє, чи є вона нижчою або вищою за середню ринкову ціну для відповідної марки, моделі та року випуску. У результаті рішення часто приймається інтуїтивно, без достатнього порівняння з іншими пропозиціями [1 - 4].

Ще одним важливим фактором є актуальність даних. Оголошення можуть залишатися на сайтах після продажу автомобіля, ціна може змінюватися, а дані можуть бути неповними або застарілими. Якщо система не має механізмів оновлення та збереження історії змін, користувач не може оцінити, як поведилася пропозиція раніше і чи є поточна ціна стабільною [1 - 4].

До проблем також належить велика кількість ручних дій. Для вибору автомобіля користувачу потрібно виконати пошук, застосувати фільтри, відкрити декілька оголошень, порівняти характеристики, перевірити ціну, зберегти посилання та повернутися до них пізніше. Такий процес є неефективним і може призводити до втрати важливих варіантів або помилкового вибору.

Отже, основними проблемами пошуку та порівняння вживаних автомобілів є розподіленість даних між різними платформами, відсутність централізованої аналітики, складність оцінювання вигідності пропозицій, нестача історичних даних про зміну цін та значні часові витрати користувача.

1.4 Необхідність централізованої платформи AutoHub

З огляду на визначені проблеми, доцільним є створення централізованої web-платформи, яка об'єднує інформацію про вживані автомобілі в одному

середовищі. Така система повинна не просто відображати оголошення, а забезпечувати користувачу повний цикл роботи з ними: пошук, фільтрацію, перегляд деталей, збереження, порівняння та аналітичну оцінку.

AutoHub розглядається як web-застосунок, спрямований на централізований моніторинг ринку вживаних автомобілів. Основна ідея платформи полягає в тому, щоб зменшити кількість ручних дій користувача та надати йому єдиний інтерфейс для роботи з пропозиціями. Завдяки цьому користувач може швидше знаходити потрібні автомобілі, порівнювати їх між собою та отримувати додаткову інформацію для прийняття рішення.

На відміну від звичайного каталогу оголошень, AutoHub має виконувати роль платформи підтримки вибору. Це означає, що система повинна не тільки зберігати дані, а й інтерпретувати їх. Наприклад, користувач має отримувати інформацію про середню ціну, історію зміни вартості, популярність певної марки або моделі та загальний рівень вигідності конкретної пропозиції.

Централізована платформа також дозволяє реалізувати єдину структуру даних. Незалежно від того, з якого джерела отримано оголошення, інформація про автомобіль зберігається у стандартизованому форматі. Це спрощує пошук, сортування, фільтрацію та порівняння автомобілів. Крім того, така структура забезпечує можливість подальшого розширення системи, наприклад додавання нових джерел, нових аналітичних показників або персональних рекомендацій.

Для адміністратора централізована система є зручною з точки зору контролю даних. Адміністративна панель дозволяє переглядати загальну статистику платформи, контролювати кількість оголошень, користувачів, записів історії цін та запускати оновлення інформації. Це робить систему не лише користувацьким сервісом, а й повноцінним інструментом керування даними.

Таким чином, необхідність створення AutoHub обумовлена потребою у зменшенні фрагментованості ринку, підвищенні зручності пошуку, автоматизації порівняння та забезпеченні аналітичної підтримки користувача під час вибору автомобіля.

1.5 Концепція AutoHub Deal Score як засобу підтримки прийняття рішення

Важливою особливістю платформи AutoHub є механізм AutoHub Deal Score. Його призначення полягає в тому, щоб допомогти користувачу не лише переглянути оголошення, а й оцінити його вигідність відносно інших пропозицій на ринку. Це дозволяє перетворити систему з простого каталогу автомобілів на інструмент підтримки прийняття рішення.

AutoHub Deal Score може бути представлений у вигляді числової оцінки за шкалою від 0 до 100 балів. Чим вищий показник, тим більш привабливою є пропозиція з точки зору ціни, пробігу, актуальності оголошення та інших параметрів. Для користувача така оцінка може відображатися у вигляді зрозумілого статусу, наприклад: «Вигідна пропозиція», «Ринкова ціна», «Дорожче ринку» або «Потребує перевірки».

Під час розрахунку Deal Score враховуються декілька факторів. Першим є співвідношення ціни конкретного автомобіля із середньою ціною подібних пропозицій. Якщо автомобіль коштує дешевше за середній рівень для відповідної марки та моделі, це підвищує його оцінку. Другим фактором є пробіг відносно року випуску: автомобіль із надмірним пробігом отримує нижчу оцінку, навіть якщо його ціна виглядає привабливою.

Також враховується свіжість оголошення. Нові оголошення можуть бути більш актуальними, тоді як дуже старі пропозиції потребують додаткової перевірки. Окремим фактором є історія зміни ціни: якщо ціна знижувалася, це може свідчити про більшу гнучкість продавця або про зміну ринкової ситуації. Додатково може враховуватися популярність марки або моделі серед інших пропозицій у системі.

Перевага Deal Score полягає в тому, що він подає складну ринкову інформацію у зрозумілому для користувача форматі. Замість самостійного аналізу десятків оголошень користувач отримує стислу оцінку, яка допомагає швидше

визначити найбільш перспективні варіанти. При цьому система не приймає рішення замість користувача, а лише надає додаткову аналітичну підказку.

Реалізація такого механізму підвищує практичну цінність AutoHub, оскільки платформа починає виконувати не тільки функцію агрегації оголошень, а й функцію інтелектуальної підтримки вибору. Це є важливою відмінністю системи від звичайних сервісів продажу автомобілів, де користувач переважно самостійно інтерпретує отриману інформацію.



Рис. 1.1 Узагальнена схема процесу моніторингу та аналізу ринку автомобілів з пробігом

Узагальнена схема процесу моніторингу та аналізу ринку автомобілів з пробігом (див. рис.1.1) відображає основну логіку роботи платформи AutoHub. Дані про автомобілі надходять із зовнішніх джерел, проходять етап обробки та зберігаються в базі даних. Після цього користувач може працювати з ними через web-інтерфейс: виконувати пошук, застосовувати фільтри, переглядати картки автомобілів, додавати пропозиції до обраного, порівнювати авто та отримувати аналітичну оцінку за допомогою Deal Score [1-4].

Отже, аналіз предметної галузі показав, що ринок автомобілів з пробігом потребує централізованого підходу до збору, обробки та представлення даних. Наявність різних джерел інформації, складність порівняння пропозицій і відсутність єдиної оцінки вигідності створюють потребу у розробці web-платформи AutoHub, яка поєднує функції каталогу, аналітичного інструмента та системи підтримки прийняття рішення.

2 АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ДЛЯ ПОШУКУ ТА МОНІТОРИНГУ АВТОМОБІЛІВ З ПРОБІГОМ

У сфері продажу та пошуку автомобілів з пробігом сьогодні використовується велика кількість web-сервісів, онлайн-маркетплейсів та спеціалізованих платформ. Основне призначення таких рішень полягає у наданні користувачам доступу до автомобільних оголошень, можливості пошуку за параметрами, перегляду характеристик транспортного засобу та зв'язку з продавцем. Водночас більшість існуючих платформ працює як окремий інформаційний ресурс і не забезпечує повного централізованого аналізу ринку [5; 6; 8 - 11].

Для користувача це означає, що пошук автомобіля часто потребує одночасного використання декількох сервісів. Наприклад, одна частина оголошень може бути розміщена на спеціалізованому автомобільному маркетплейсі, інша — на загальному сайті оголошень, а ще частина — на міжнародних ресурсах. Через це користувач змушений самотійно порівнювати ціни, пробіг, рік випуску, регіон продажу та інші параметри, що збільшує час пошуку і ускладнює прийняття рішення.

Для аналізу було обрано чотири поширені web-сервіси, які використовуються для пошуку та перегляду автомобільних оголошень: AUTO.RIA [8], OLX Авто [9], mobile.de [10] та AutoScout24 [11]. Ці платформи мають різну цільову аудиторію, різну географічну орієнтацію та різний рівень аналітичної підтримки користувача, тому їх порівняння дозволяє визначити функціональні обмеження існуючих рішень і обґрунтувати необхідність розробки AutoHub.

2.1 Огляд сервісу AUTO.RIA

AUTO.RIA є одним із найбільш відомих українських сервісів для купівлі та продажу автомобілів [8]. Платформа орієнтована на український ринок і містить велику кількість оголошень про нові та вживані автомобілі. Користувач може здійснювати пошук за маркою, моделлю, ціною, роком випуску, типом кузова, пробігом, регіоном та іншими характеристиками.

Важливою перевагою AUTO.RIA є спеціалізація саме на автомобільному сегменті. На відміну від загальних платформ оголошень, сервіс має більш деталізовану структуру картки автомобіля, розширені фільтри та додаткові інструменти для перегляду інформації про транспортний засіб. Це робить платформу зручною для користувачів, які шукають автомобіль саме на українському ринку (див. рис. 2.1).

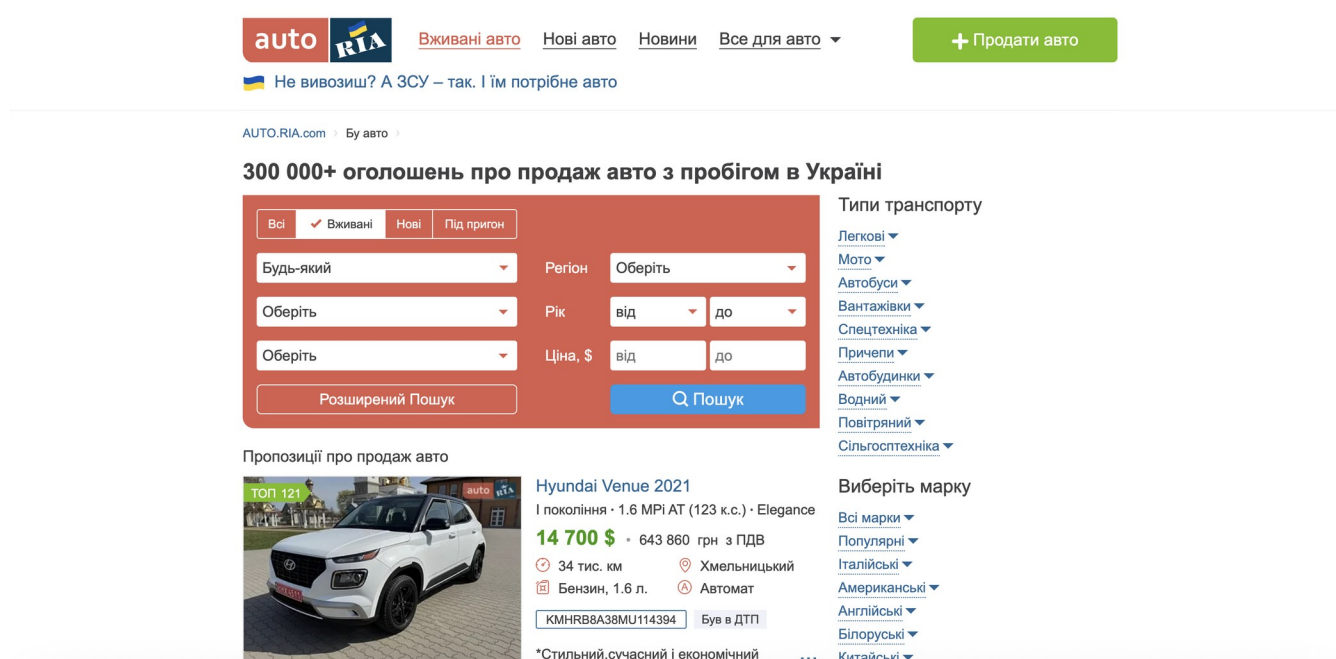


Рис. 2.1 Приклад інтерфейсу сервісу AUTO.RIA

До переваг AUTO.RIA можна віднести велику кількість оголошень, орієнтацію на українських користувачів, зручний пошук за параметрами,

наявність фільтрів і детальний опис автомобілів. Користувач може швидко переглянути базові характеристики авто та перейти до повної картки оголошення.

Водночас сервіс має обмеження з точки зору централізованого моніторингу. AUTO.RIA працює переважно з власною базою оголошень, тому користувач не отримує повної картини ринку з різних платформ. Також сервіс не надає прозорої інтегральної оцінки вигідності конкретної пропозиції, через що користувач повинен самостійно визначати, чи відповідає ціна автомобіля його стану, пробігу та середньоринковим умовам.

2.2 Огляд сервісу OLX Авто

OLX Авто є розділом загальної платформи оголошень OLX, який призначений для розміщення пропозицій щодо продажу легкових автомобілів [9]. Сервіс має широку аудиторію, оскільки OLX використовується для різних категорій товарів і послуг. Завдяки цьому в розділі автомобілів можна знайти значну кількість приватних оголошень.

Перевагою OLX Авто є простота використання. Користувач може швидко відкрити розділ автомобілів, обрати регіон, встановити ціновий діапазон і переглянути доступні пропозиції (див. рис. 2.2). Платформа є зручною для продавців, оскільки дозволяє швидко створити оголошення без складної структури заповнення.

До переваг OLX Авто належать доступність, простий інтерфейс, велика кількість приватних оголошень та зручна комунікація між покупцем і продавцем. Сервіс підходить для швидкого перегляду пропозицій у певному регіоні або ціновому діапазоні.

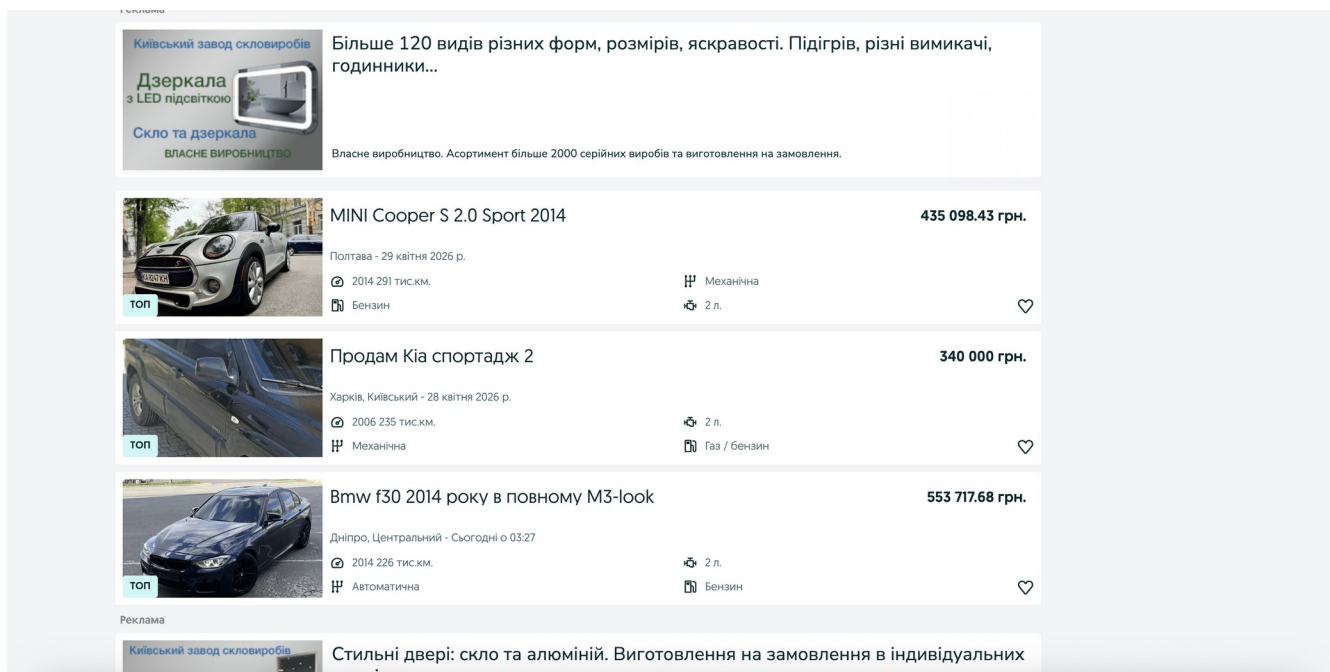


Рис. 2.2 Приклад інтерфейсу сервісу OLX Авто

Однак OLX Авто не є вузькоспеціалізованою автомобільною платформою. Через це функціональність аналізу автомобілів є обмеженою. У порівнянні зі спеціалізованими сервісами, тут менше можливостей для глибокої фільтрації, перевірки характеристик, порівняння автомобілів та аналізу ринку. Користувачу складніше оцінити реальну вигідність пропозиції, оскільки сервіс не надає системної аналітики та історії зміни цін.

2.3 Огляд сервісу mobile.de

mobile.de є одним із найбільших європейських онлайн-майданчиків для купівлі та продажу автомобілів. Сервіс орієнтований переважно на європейський ринок і містить оголошення про нові та вживані автомобілі, мотоцикли, комерційний транспорт та інші транспортні засоби [10].

Платформа має розвинену систему пошуку та фільтрації. Користувач може обирати марку, модель, рік випуску, пробіг, тип пального, трансмісію, ціну, країну або регіон продажу (див. рис. 2.3). Також сервіс надає інформацію про

дилерів, фінансові умови, лізинг та інші додаткові параметри, що робить його корисним для користувачів, які розглядають купівлю автомобіля в Європі.

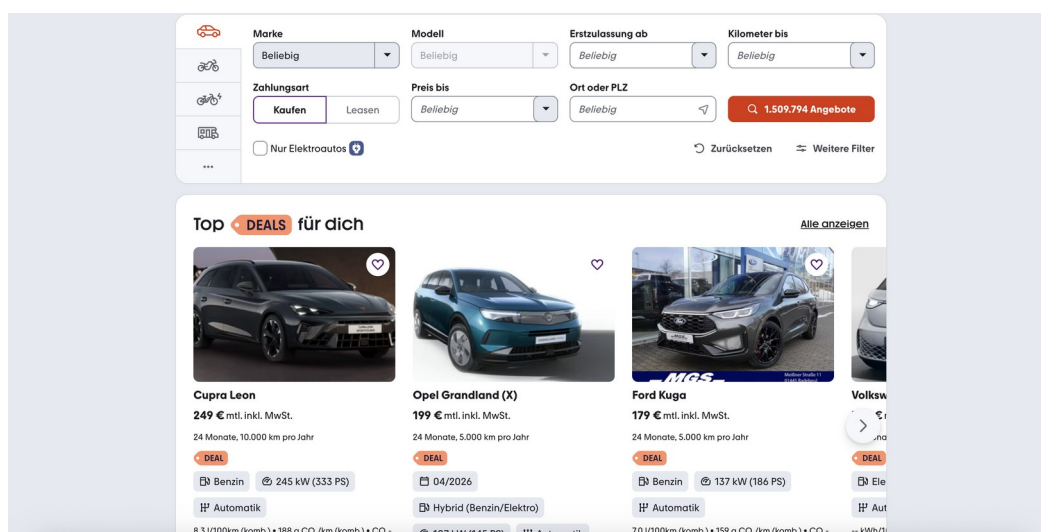


Рис. 2.3 Приклад інтерфейсу сервісу mobile.de

Перевагами mobile.de є велика база оголошень, зручний пошук, якісна деталізація характеристик автомобіля та орієнтація на європейський ринок. Сервіс дозволяє переглядати широкий спектр пропозицій і є корисним для аналізу цін на автомобілі за межами України.

Недоліком mobile.de для українського користувача є те, що сервіс не орієнтований безпосередньо на український ринок. Дані з цієї платформи потребують додаткової інтерпретації з урахуванням витрат на доставку, розмитнення, реєстрацію та інших факторів. Також mobile.de не забезпечує централізованого поєднання даних з українських платформ і не надає користувачу єдиної оцінки вигідності пропозиції в контексті локального ринку.

2.4 Огляд сервісу AutoScout24

AutoScout24 є ще однією відомою європейською платформою для пошуку нових і вживаних автомобілів [11]. Сервіс використовується в багатьох країнах Європи та дозволяє переглядати оголошення за маркою, моделлю, типом кузова, ціною, пробігом, роком випуску, типом пального та іншими параметрами.

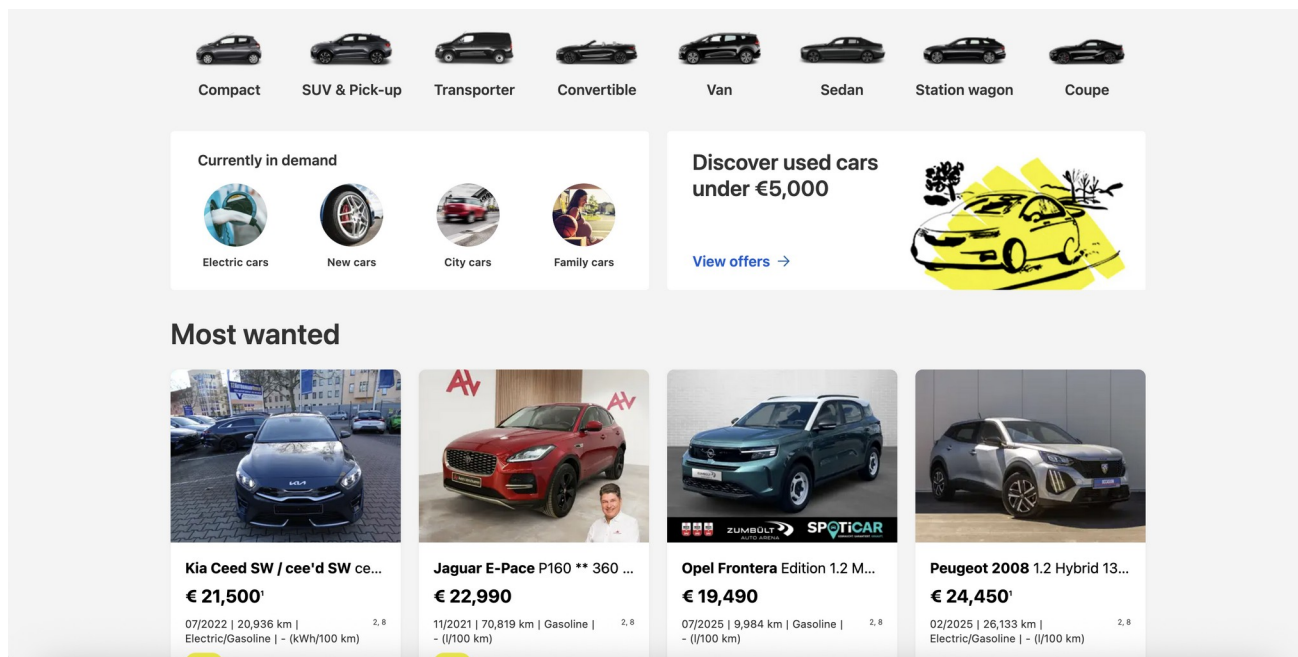


Рис. 2.4 Приклад інтерфейсу сервісу AutoScout24

Платформа має зручний web-інтерфейс і орієнтована як на приватних користувачів, так і на дилерів (див. рис. 2.4). AutoScout24 дозволяє переглядати детальні картки автомобілів, фото, опис, технічні параметри та контактну інформацію продавця. Також сервіс підтримує різні категорії транспортних засобів, зокрема електромобілі, гібридні авто, позашляховики та мотоцикли.

Перевагами AutoScout24 є широка база європейських оголошень, деталізований пошук, якісна структура картки автомобіля та можливість аналізу пропозицій у межах міжнародного ринку. Сервіс є корисним для користувачів, які планують купівлю автомобіля за кордоном або хочуть порівняти ціни на різних європейських ринках.

Основним обмеженням AutoScout24 є недостатня адаптація до потреб українського користувача. Платформа не забезпечує повної інтеграції з українськими джерелами оголошень, не враховує локальні особливості ринку та не надає персоналізованої оцінки вигідності автомобіля. Як і інші розглянуті сервіси, AutoScout24 переважно надає дані, але не виконує функцію комплексної підтримки прийняття рішення.

2.5 Порівняльний аналіз функціональних можливостей аналогів

Для узагальнення результатів аналізу було сформовано порівняльну таблицю, у якій розглянуто ключові можливості існуючих сервісів та web-платформи AutoHub (див. табл. 2.1). Основну увагу приділено тим функціям, які мають значення для централізованого моніторингу, пошуку, порівняння та аналітичного оцінювання автомобільних оголошень.

AutoHub відрізняється від аналогів тим, що орієнтований не лише на перегляд оголошень, а й на їхню аналітичну обробку. Платформа має поєднувати каталог, фільтрацію, обране, порівняння, історію зміни цін та механізм Deal Score, який допомагає користувачу швидше оцінити якість конкретного варіанту.

2.6 Виявлені недоліки існуючих систем

У результаті аналізу існуючих програмних рішень було визначено низку недоліків, які ускладнюють процес пошуку та вибору вживаного автомобіля. Насамперед більшість платформ працює в межах власної бази оголошень. Це означає, що користувач отримує лише частину інформації про ринок і змушений додатково переглядати інші сервіси.

Другим важливим недоліком є недостатня аналітична підтримка. Більшість сервісів дозволяє знайти автомобіль за заданими параметрами, однак не пояснює, наскільки конкретна пропозиція є вигідною у порівнянні з ринком.

Також існуючі системи не завжди надають можливість повноцінного порівняння автомобілів. Якщо така функція присутня, вона часто реалізована частково або не враховує аналітичні показники.

Окремою проблемою є відсутність прозорості історії зміни ціни. Для прийняття рішення важливо розуміти, чи змінювалася вартість автомобіля, чи знижував продавець ціну, як давно оголошення було опубліковане та чи залишається воно актуальним.

Таким чином, аналіз існуючих систем показав необхідність створення web-платформи, яка поєднує функції каталогу, централізованого моніторингу, порівняння, аналітики та оцінювання вигідності пропозицій. Саме ці завдання покладено в основу розробки AutoHub.

Таблиця 2.1

Порівняльна характеристика web-сервісів для пошуку та моніторингу
вживаних автомобілів

Показник	AUTO.RI A	OLX Авто	mobile.de	AutoScout24	AutoHub
Основне призначення	Купівля та продаж авто	Загальний сервіс оголошень з розділом авто	Європейський автобазар	Європейський маркетплейс авто	Централізований моніторинг, порівняння та аналітика ринку вживаних авто
Орієнтація на український ринок	+	+	-	-	+
Пошук за маркою і моделлю	+	+	+	+	+
Фільтрація за ціною	+	+	+	+	+
Фільтрація за пробігом	+	+	+	+	+
Фільтрація за роком випуску	+	+	+	+	+
Порівняння автомобілів	Частково	-	Частково	Частково	+
Аналітичні можливості	Частково	-	Частково	Частково	+
Централізований збір з різних платформ	-	-	-	-	+
Оцінювання вигідності пропозиції Deal Score	-	-	-	-	+
Історія зміни цін	Частково	-	Частково	Частково	+

3 ПОСТАНОВКА ЗАДАЧІ ТА ФОРМУВАННЯ ВИМОГ ДО WEB-ПЛАТФОРМИ AUTONUB

Розробка web-платформи AutoHub базується на результатах аналізу предметної галузі та існуючих програмних рішень для пошуку, перегляду й моніторингу вживаних автомобілів. У попередніх розділах було встановлено, що сучасний ринок автомобілів з пробігом є інформаційно розподіленим: оголошення розміщуються на різних онлайн-платформах, мають різну структуру, різний рівень деталізації та не завжди дозволяють користувачу швидко порівняти кілька пропозицій між собою.

Основна складність полягає не лише у пошуку автомобіля за певними параметрами, а й у подальшій інтерпретації отриманих результатів. Користувач повинен самостійно оцінювати, чи є конкретна ціна ринковою, чи відповідає пробіг року випуску, наскільки свіже оголошення, чи змінювалася ціна раніше та чи варто розглядати автомобіль як потенційно вигідну пропозицію. У більшості існуючих сервісів ця робота виконується вручну, без достатньої аналітичної підтримки.

У зв'язку з цим виникає потреба у створенні web-платформи, яка не тільки зберігає та відображає автомобільні оголошення, а й допомагає користувачу приймати рішення. Саме тому AutoHub розглядається як система централізованого моніторингу ринку вживаних автомобілів з елементами аналітичної підтримки, персоналізації та оцінювання вигідності конкретних пропозицій.

Платформа повинна забезпечувати взаємодію трьох основних категорій користувачів: гостей, зареєстрованих користувачів та адміністраторів. Для кожної ролі визначається власний набір функцій і рівень доступу до можливостей системи. Це дозволяє розмежувати публічний перегляд даних, персональні можливості користувача та адміністративне керування платформою.

Окремою особливістю проєкту є механізм **AutoHub Deal Score**. Його завдання полягає у формуванні узагальненої оцінки вигідності автомобільного оголошення. Така оцінка дозволяє користувачу не тільки бачити параметри автомобіля, а й швидко розуміти, наскільки конкретна пропозиція є привабливою відносно інших варіантів на ринку.

3.1 Постановка задачі розробки

Задача розробки полягає у створенні web-платформи AutoHub, призначеної для централізованого моніторингу ринку автомобілів з пробігів, пошуку пропозицій за заданими параметрами, порівняння автомобілів, збереження обраних оголошень, перегляду аналітичних даних та оцінювання вигідності пропозицій.

Система повинна забезпечувати зручну роботу з автомобільними оголошеннями в одному середовищі. Це означає, що користувачу не потрібно переходити між кількома сайтами, самостійно зберігати посилання, вручну порівнювати параметри автомобілів або вести окремі таблиці для аналізу. AutoHub має об'єднати ці дії в межах одного web-застосунку.

З технічної точки зору система повинна бути реалізована як клієнт-серверний web-застосунок. Клієнтська частина відповідає за інтерфейс користувача, відображення каталогу, сторінок деталей, обраного, порівняння, аналітики, профілю та адміністративної панелі. Серверна частина відповідає за обробку запитів, реалізацію бізнес-логіки, роботу з базою даних, авторизацію користувачів, розмежування доступу та розрахунок аналітичних показників [7].

Основними задачами, які має вирішувати AutoHub, є:

- централізоване зберігання даних про вживані автомобілі;
- надання користувачу каталогу автомобільних оголошень;
- реалізація пошуку за ключовими параметрами автомобіля;
- фільтрація оголошень за технічними та ринковими характеристиками;
- перегляд детальної інформації про автомобіль;

- збереження цікавих пропозицій у списку обраного;
- порівняння декількох автомобілів за основними характеристиками;
- відображення аналітичної інформації про ринок;
- збереження та перегляд історії зміни цін;
- розрахунок показника AutoHub Deal Score;
- підтримка ролей користувачів і адміністраторів;
- забезпечення адміністративного керування даними.

Важливо, що платформа не повинна обмежуватися лише функціями каталогу. Якщо система просто відображає список автомобілів, вона повторює базові можливості вже існуючих сервісів. Тому основний акцент у розробці робиться на поєднанні каталогу, персональних дій користувача, аналітики та оцінювання вигідності пропозицій.

З боку звичайного користувача система має спростити процес вибору автомобіля. Користувач повинен мати можливість швидко знайти потрібні пропозиції, відкрити детальну картку, додати автомобіль до обраного, порівняти його з іншими варіантами та отримати коротку аналітичну оцінку. Це дозволяє зменшити кількість ручних операцій і підвищити якість прийняття рішення.

З боку адміністратора система має забезпечувати контроль за даними та станом платформи. Адміністратор повинен мати змогу переглядати статистику, контролювати кількість автомобілів, користувачів, обраних оголошень, записів історії цін, а також запускати оновлення даних. Це дозволяє підтримувати актуальність інформації та контролювати роботу системи [7].

Таким чином, постановка задачі полягає у розробці web-платформи, яка об'єднує функції інформаційної системи, каталогу автомобілів, аналітичного інструмента та засобу підтримки прийняття рішення щодо купівлі вживаного автомобіля.

3.2 Функціональні вимоги

Функціональні вимоги визначають перелік дій, які система повинна виконувати для користувача. У випадку AutoHub ці вимоги формуються на основі потреб різних категорій користувачів: гостей, зареєстрованих користувачів і адміністраторів. Кожна з цих ролей має власний набір сценаріїв взаємодії з платформою [12; 13].

Функціональні вимоги до AutoHub можна поділити на кілька груп: вимоги до роботи з обліковими записами, вимоги до каталогу автомобілів, вимоги до персональних функцій користувача, вимоги до аналітики, вимоги до адміністративної частини та вимоги до механізму AutoHub Deal Score [1 - 4].

До функціональних вимог належать:

1. Збір і відображення оголошень з різних сайтів.
2. Пошук і розширена фільтрація за параметрами авто.
3. Порівняння кількох пропозицій.
4. Відображення історії перевірки та ключових атрибутів.
5. Аналітика цін і виявлення схожих оголошень автомобілів за VIN кодом.
6. Збереження обраних варіантів і сповіщення про нові пропозиції.

Функція реєстрації та авторизації необхідна для створення персонального середовища користувача. Неавторизований користувач може переглядати загальнодоступні сторінки, однак для роботи з обраним, профілем і персональними діями потрібна авторизація. Розмежування доступу дозволяє відокремити звичайного користувача від адміністратора та захистити службові функції системи [7].

Каталог автомобілів є базовим елементом платформи. Він повинен містити список доступних оголошень із короткою інформацією про кожне авто: марку, модель, рік випуску, ціну, пробіг, регіон, платформу-джерело та основні технічні характеристики. Каталог повинен підтримувати пошук і фільтрацію, оскільки без цих можливостей користувачу буде складно працювати з великою кількістю пропозицій.

Сторінка детальної картки автомобіля повинна надавати повнішу інформацію про конкретне оголошення. До такої інформації належать фото автомобіля, технічні характеристики, опис, ціна, історія зміни ціни, дата публікації, дата оновлення та посилання на джерело. Саме на цій сторінці користувач приймає попереднє рішення щодо того, чи варто розглядати автомобіль далі.

Функція обраного дозволяє користувачу зберігати цікаві пропозиції для подальшого аналізу. Це важливо, оскільки процес вибору автомобіля зазвичай не завершується одним переглядом каталогу. Користувач може повертатися до збережених варіантів, порівнювати їх між собою та спостерігати за зміною ціни.

Функція порівняння дозволяє одночасно аналізувати кілька автомобілів. Порівняння має відображати ключові характеристики у зручному форматі, щоб користувач міг швидко побачити різницю між варіантами. Це зменшує кількість ручних дій і робить процес вибору більш структурованим.

Аналітичний модуль повинен відображати узагальнені показники ринку. До таких показників належать кількість оголошень, середні ціни, популярні марки, розподіл автомобілів за роками випуску, цінові діапазони та інші дані. Така інформація допомагає користувачу краще орієнтуватися у ринку, а не оцінювати кожне оголошення ізольовано.

Механізм AutoHub Deal Score є окремою функціональною вимогою, оскільки саме він формує відмінність системи від звичайного каталогу. Deal Score повинен розраховувати узагальнену оцінку вигідності автомобіля на основі кількох факторів: ціни відносно середнього ринку, пробігу, року випуску, актуальності оголошення, історії зміни ціни та популярності моделі. Результат має бути зрозумілим для користувача та подаватися у вигляді числового показника або короткої текстової категорії.

Адміністративна частина повинна забезпечувати керування даними системи. Адміністратор має переглядати статистику, контролювати кількість користувачів, оголошень, записів історії цін, запускати оновлення даних і мати

доступ до службових функцій. Це необхідно для стабільної роботи платформи та підтримки актуальності інформації.

3.4 Нефункціональні вимоги

Нефункціональні вимоги описують не конкретні функції системи, а її якісні характеристики. Вони визначають, наскільки швидко, безпечно, стабільно та зручно повинна працювати web-платформа AutoHub. Для такого типу системи особливо важливими є швидкодія, захист даних, адаптивність інтерфейсу, масштабованість і надійність [12; 13].

До нефункціональних вимог належать:

1. Час завантаження сторінок — не більше 3 секунд.
2. Час відповіді основних API-запитів — до 1 секунди для вибірки до 100 авто.
3. Підтримка щонайменше 50 одночасних користувачів.
4. Зберігання не менше 10 000 оголошень у базі даних.
5. Зберігання паролів у хешованому вигляді з використанням BCrypt, доступ до захищених сторінок — через JWT.
6. Коректна робота інтерфейсу на екранах від 360 px та в основних браузерях: Chrome, Safari, Firefox, Edge.

Вимога до часу завантаження сторінок пов'язана з тим, що користувач очікує швидкої взаємодії із системою. Якщо каталог, аналітика або сторінка деталей завантажуються занадто довго, це погіршує користувацький досвід і знижує практичну цінність платформи. Тому для основних сторінок встановлюється обмеження часу завантаження до 3 секунд.

Вимога до часу відповіді API є важливою для роботи з каталогом, фільтрами, обраним, деталями автомобіля та аналітикою. Основні запити повинні оброблятися швидко, оскільки користувач часто змінює параметри фільтрації та очікує оперативного оновлення результатів. Для типових вибірок до 100 автомобілів час відповіді серверної частини не повинен перевищувати 1 секунди.

Підтримка щонайменше 50 одночасних користувачів визначає мінімальний рівень навантаження, за якого система повинна працювати стабільно. Такий показник є достатнім для демонстраційної та навчальної версії web-платформи, але архітектура системи повинна допускати подальше масштабування [12; 13].

Вимога до зберігання не менше 10 000 оголошень пов'язана з тим, що ринок автомобілів з пробігом містить значну кількість пропозицій. Навіть у межах демонстраційної версії база даних повинна бути спроектована так, щоб підтримувати велику кількість записів і забезпечувати ефективний пошук за індексованими полями [14; 15; 16].

Безпека є критично важливою характеристикою системи, оскільки AutoHub працює з обліковими записами користувачів. Паролі повинні зберігатися виключно у хешованому вигляді. Для цього використовується BCrypt. Доступ до захищених ресурсів має здійснюватися через JWT-токени, що дозволяє перевіряти автентичність користувача та його роль без зберігання стану сесії на сервері [7].

Адаптивність інтерфейсу необхідна через те, що користувачі можуть працювати з платформою з різних пристроїв. Інтерфейс повинен коректно відображатися як на мобільних екранах шириною від 360 px, так і на ноутбуках або десктопах. Також система повинна підтримувати актуальні версії основних браузерів: Chrome, Safari, Firefox та Edge.

Нефункціональні вимоги також передбачають зрозумілу обробку помилок. У випадку недоступності сервера, помилки авторизації, відсутності даних або некоректного запиту користувач повинен отримувати зрозуміле повідомлення. Це дозволяє знизити кількість ситуацій, у яких користувач не розуміє причину помилки або не може продовжити роботу [7].

3.5 Ролі користувачів та сценарії доступу

У web-платформі AutoHub передбачено три основні ролі користувачів: гість, зареєстрований користувач і адміністратор. Така модель доступу дозволяє розділити публічні, персональні та службові функції системи. Розмежування

ролей є важливим як з точки зору безпеки, так і з точки зору організації інтерфейсу.

Гість — це користувач, який ще не пройшов авторизацію. Для нього система повинна бути доступною у режимі ознайомлення. Гість може переглядати головну сторінку, відкривати каталог автомобілів, використовувати базові фільтри, переглядати сторінки деталей авто, знайомитися з аналітикою та перейти до реєстрації або входу. Водночас гість не може додавати автомобілі до обраного, користуватися персональним профілем або відкривати адміністративну панель [7].

Зареєстрований користувач отримує доступ до персональних функцій. Він може виконувати всі дії гостя, а також додавати автомобілі до обраного, переглядати список збережених авто, порівнювати кілька пропозицій, відкривати профіль, переглядати персональну інформацію та взаємодіяти з аналітичними функціями системи. Саме ця роль є основною для сценарію вибору автомобіля [7].

Адміністратор має найширші права доступу. Окрім можливостей звичайного користувача, адміністратор може переглядати адміністративну панель, бачити статистику системи, переглядати користувачів, контролювати оголошення, запускати оновлення даних і керувати службовими процесами. Адміністративна роль необхідна для підтримки системи в актуальному стані та контролю її роботи [7].

Сценарії доступу можна описати таким чином. Якщо користувач заходить на сайт без авторизації, він працює як гість і має доступ лише до публічної частини платформи. Після реєстрації або входу система визначає роль користувача та відкриває відповідні розділи. Якщо роль користувача USER, йому доступні персональні функції. Якщо роль ADMIN, додатково відкривається адміністративна панель.

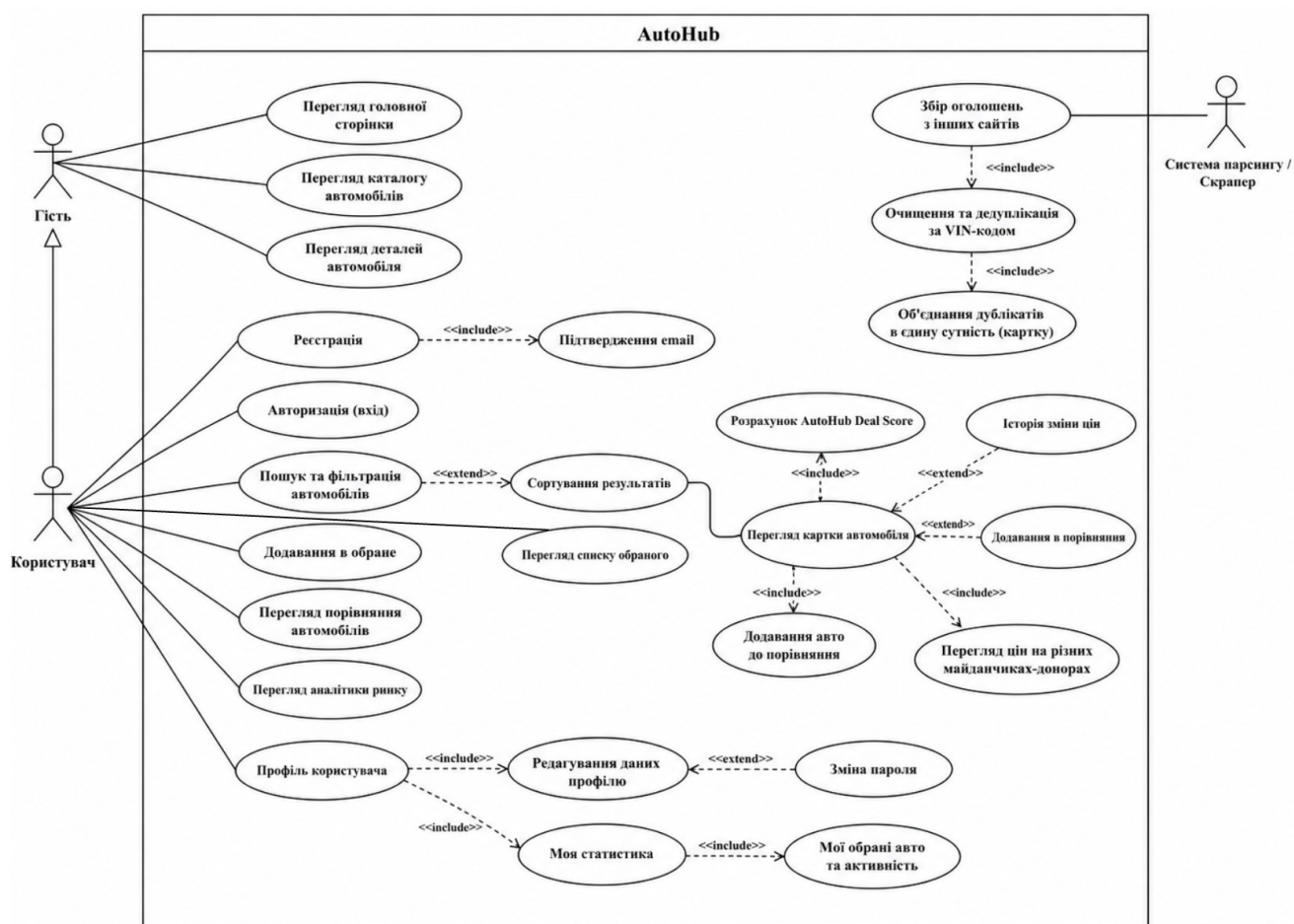


Рис. 3.1 Діаграма варіантів використання web-платформи AutoHub

Діаграма варіантів використання відображає основні сценарії взаємодії користувачів із платформою AutoHub (див. рис. 3.1). Гість має доступ до перегляду публічних сторінок, каталогу та деталей автомобіля. Зареєстрований користувач може додавати автомобілі до обраного, порівнювати їх і працювати з профілем. Адміністратор отримує доступ до службових функцій: перегляду статистики, керування користувачами, оголошеннями та запуску оновлення даних.

Така модель доступу дозволяє зробити систему логічно структурованою. Публічна частина забезпечує ознайомлення з платформою, користувацька частина — персональну роботу з автомобілями, а адміністративна частина — контроль і підтримку системи.

4 ПРОЄКТУВАННЯ WEB-ПЛАТФОРМИ AUTOHUB

Проєктування web-платформи AutoHub виконується на основі сформованих функціональних і нефункціональних вимог. На цьому етапі визначається загальна структура системи, взаємодія між клієнтською та серверною частинами, логіка роботи з базою даних, принципи авторизації користувачів, структура основних сутностей і алгоритм розрахунку оцінки вигідності автомобільного оголошення [7].

Оскільки AutoHub призначений для централізованого моніторингу ринку вживаних автомобілів, система повинна забезпечувати роботу з різними типами даних: інформацією про автомобілі, користувачів, обрані оголошення, історію зміни цін, аналітичні показники та результати розрахунку Deal Score. Для цього доцільно використати клієнт-серверну архітектуру, у якій frontend відповідає за взаємодію з користувачем, backend — за бізнес-логіку та обробку запитів, а база даних — за зберігання структурованої інформації [14; 15; 16].

Проєктування системи також передбачає розмежування доступу між різними ролями користувачів. Гість отримує доступ до публічної частини платформи, зареєстрований користувач — до персональних функцій, а адміністратор — до службових можливостей керування системою. Такий підхід дозволяє зробити web-платформу безпечнішою, логічно структурованою та зручною для подальшого розширення.

4.1 Загальна архітектура системи

Загальна архітектура web-платформи AutoHub побудована за клієнт-серверним принципом. Користувач взаємодіє із системою через web-інтерфейс, реалізований на базі React. Клієнтська частина надсилає HTTP-запити до серверної частини через REST API. Серверна частина, реалізована за допомогою

Java та Spring Boot, обробляє ці запити, виконує бізнес-логіку, звертається до бази даних PostgreSQL та повертає результат у форматі JSON [19-25].

У загальному вигляді система складається з таких основних компонентів:

- клієнтська частина web-застосунку;
- серверна частина;
- модуль авторизації та автентифікації;
- модуль роботи з автомобілями;
- модуль обраного;
- модуль порівняння автомобілів;
- аналітичний модуль;
- модуль AutoHub Deal Score;
- адміністративний модуль;
- база даних PostgreSQL;
- засоби керування міграціями бази даних;
- середовище контейнеризованого запуску.

Клієнтська частина відповідає за відображення сторінок web-застосунку. До таких сторінок належать головна сторінка, каталог автомобілів, сторінка деталей оголошення, сторінка обраного, порівняння автомобілів, аналітика, профіль користувача, сторінки входу та реєстрації, а також адміністративна панель. Основне завдання frontend-частини полягає у забезпеченні зручної взаємодії користувача із системою [21; 22].

Серверна частина виконує обробку запитів, перевірку прав доступу, взаємодію з базою даних, формування відповідей для клієнта, розрахунок аналітичних показників та оцінки Deal Score. Backend також відповідає за безпеку системи, оскільки саме на сервері перевіряються JWT-токени, права користувачів і доступ до захищених ресурсів [7].

База даних PostgreSQL використовується для зберігання структурованої інформації. У ній зберігаються дані про користувачів, автомобілі, зображення автомобілів, обрані оголошення та історію зміни цін. Така структура дозволяє

підтримувати основну функціональність платформи та забезпечує можливість подальшого розширення системи [14 - 16].

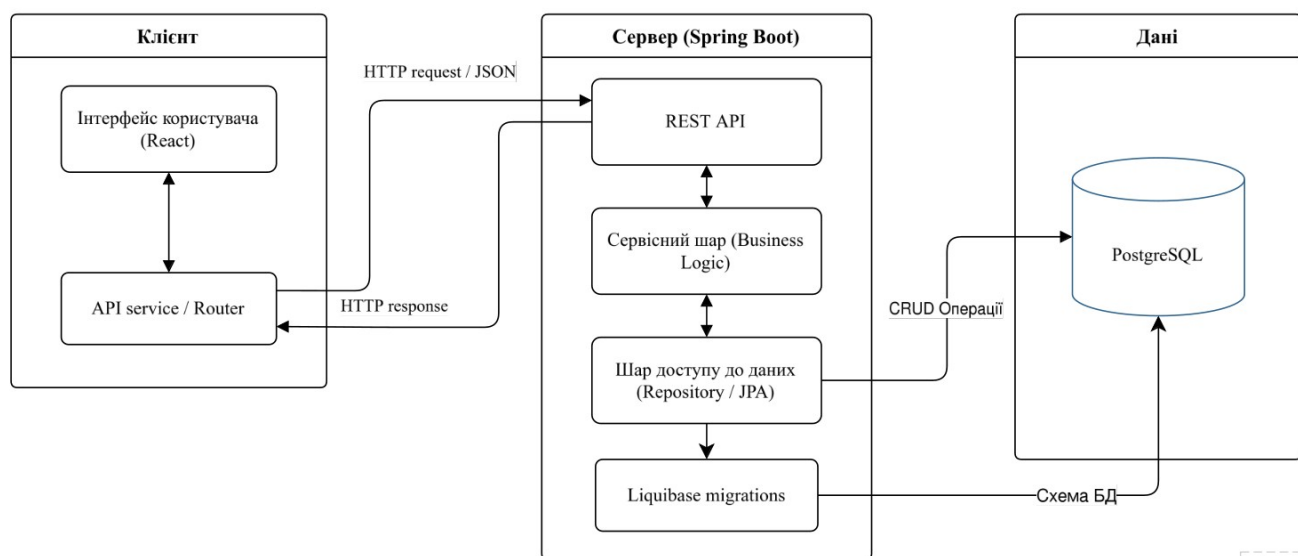


Рис. 4.1 Загальна архітектура web-платформи AutoHub

Загальна архітектура AutoHub (див. рис. 4.1) відображає взаємодію користувача з web-інтерфейсом, серверною частиною, базою даних та зовнішніми джерелами автомобільних оголошень. Користувач надсилає запити через frontend-частину, backend обробляє їх, звертається до бази даних або сервісних модулів і повертає результат. Окремо виділяються модулі авторизації, аналітики, Deal Score та адміністративного керування, які забезпечують розширену функціональність платформи [21; 22].

4.2 Архітектура клієнтської та серверної частини

Архітектура AutoHub передбачає чітке розділення клієнтської та серверної частин. Такий підхід є доцільним для сучасних web-застосунків, оскільки дозволяє окремо розробляти, тестувати та масштабувати frontend і backend. Крім того, розділення логіки інтерфейсу та бізнес-логіки спрощує підтримку проєкту і

дозволяє в майбутньому підключати інші клієнтські застосунки, наприклад мобільний додаток [21; 22].

Клієнтська частина реалізується за допомогою React. Компонентний підхід React дозволяє будувати інтерфейс із незалежних частин: карток автомобілів, форм фільтрації, таблиць порівняння, блоків аналітики, навігаційного меню, форм входу та реєстрації. Така структура є зручною для повторного використання компонентів на різних сторінках [21; 22].

Для швидкого запуску та збирання frontend-частини використовується Vite. Він забезпечує швидку розробку, оновлення сторінок під час зміни коду та формування production-збірки застосунку. У межах AutoHub frontend відповідає за маршрутизацію між сторінками, взаємодію з REST API, збереження токена авторизації, обробку дій користувача та відображення отриманих від сервера даних.

Серверна частина реалізується на основі Java та Spring Boot. Вона має багат шарову структуру, що включає контролери, сервіси, репозиторії, DTO-об'єкти, мапери, сутності та конфігураційні класи. Такий підхід дозволяє розділити відповідальність між різними частинами backend-логіки [7; 14; 19; 20].

Контролери відповідають за приймання HTTP-запитів і повернення відповідей. Наприклад, CarController обробляє запити до каталогу автомобілів, FavoriteController — дії з обраним, AnalyticsController — запити до аналітики, AuthController — реєстрацію та авторизацію користувачів. Контролери не повинні містити складної бізнес-логіки, вони лише передають запити до відповідних сервісів [7].

Сервісний шар відповідає за основну бізнес-логіку системи. Наприклад, CarService виконує пошук, фільтрацію та отримання деталей автомобілів, FavoriteService працює з обраними оголошеннями, AnalyticsService формує

аналітичні показники, DealScoreService розраховує оцінку вигідності автомобіля. Такий розподіл робить код більш структурованим і спрощує тестування.

Репозиторії відповідають за доступ до бази даних. Для цього використовується Spring Data JPA, що дозволяє працювати з таблицями бази через Java-інтерфейси. Наприклад, CarRepository використовується для роботи з автомобілями, UserRepository — з користувачами, FavoriteRepository — з обраними авто, PriceHistoryRepository — з історією цін [19; 20].

Обмін даними між frontend і backend відбувається через REST API. Дані передаються у форматі JSON, що є зручним для web-застосунків. Для того щоб не передавати напямку внутрішні сутності бази даних, використовуються DTO-об'єкти. Це дозволяє контролювати структуру відповідей, приховувати службові поля та зменшувати залежність між базою даних і клієнтською частиною [21; 22].

Безпека системи реалізується за допомогою Spring Security та JWT. Після успішної авторизації користувач отримує токен, який надалі додається до запитів до захищених ресурсів. Сервер перевіряє токен, визначає користувача та його роль. Це дозволяє розмежувати доступ до функцій USER і ADMIN.

Liquibase використовується для керування змінами структури бази даних. Усі таблиці та зміни схеми описуються у вигляді міграцій. Це дозволяє відтворювати структуру бази даних у різних середовищах та зберігати історію змін [16].

Docker і Docker Compose застосовуються для контейнеризованого запуску backend-частини та бази даних. Це спрощує налаштування середовища, оскільки всі необхідні сервіси можна запустити однією командою. Такий підхід також робить проєкт більш переносимим [23].

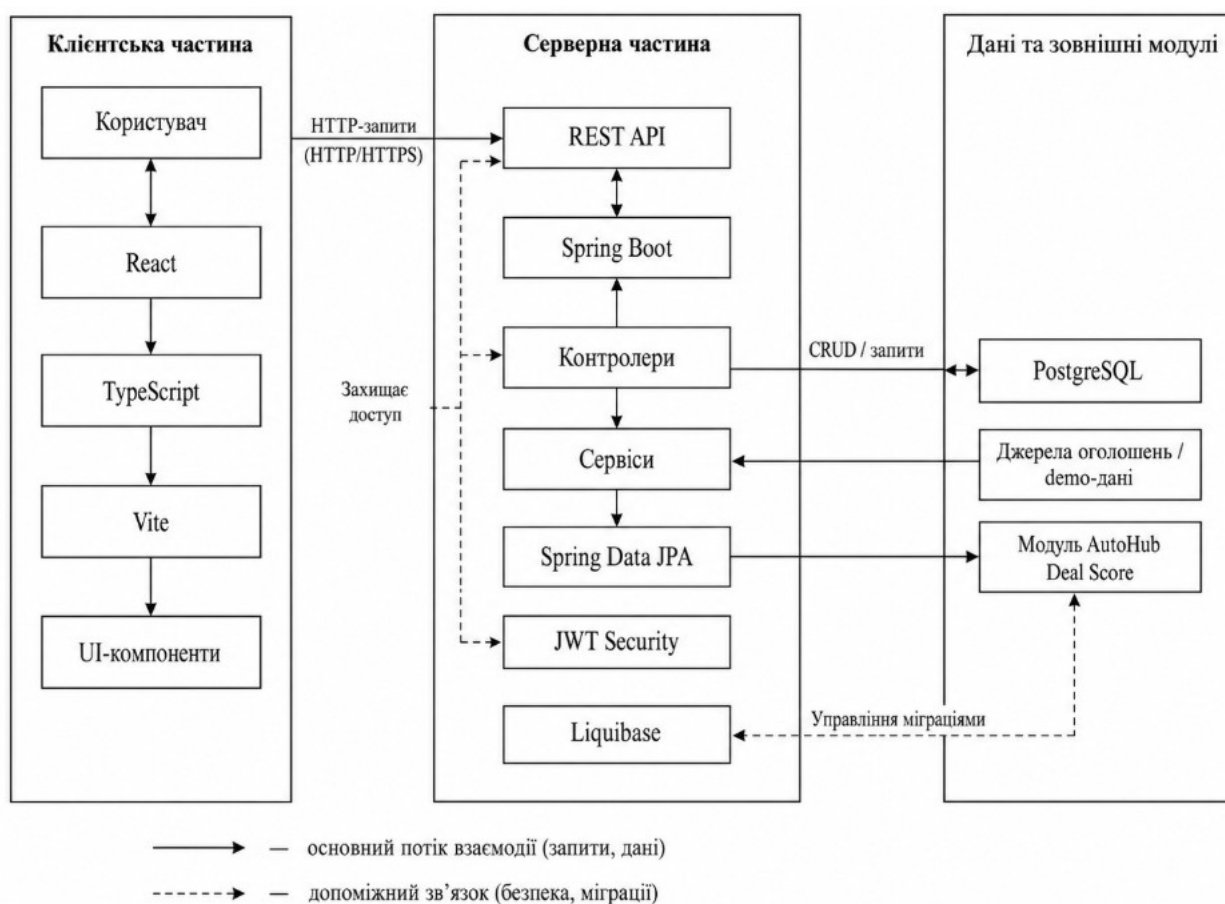


Рис. 4.2 Діаграма взаємодії технологій проєкту

Діаграма взаємодії технологій (див. рис. 4.2) показує, як основні інструменти розробки пов'язані між собою. React формує клієнтський інтерфейс і звертається до REST API. Spring Boot обробляє запити, Spring Security перевіряє доступ, Spring Data JPA забезпечує взаємодію з PostgreSQL, Liquibase керує структурою бази даних, а Docker Compose відповідає за запуск сервісів у контейнерах.

4.3 Проєктування бази даних

База даних AutoHub призначена для зберігання основної інформації, необхідної для роботи платформи. Оскільки система працює з автомобільними оголошеннями, користувачами, обраними пропозиціями та історією цін, структура бази даних повинна забезпечувати зв'язки між цими сутностями [14 - 16].

У проєкті використовується реляційна база даних PostgreSQL. Вибір реляційної моделі є доцільним, оскільки дані мають чітку структуру та логічні зв'язки. Наприклад, один користувач може мати багато обраних автомобілів, один автомобіль може мати багато зображень, один автомобіль може мати багато записів історії цін. Такі зв'язки зручно реалізовувати за допомогою первинних і зовнішніх ключів [14 - 16].

База даних AutoHub складається з п'яти основних таблиць:

- users;
- cars;
- car_images;
- favorites;
- price_history.

Таблиця users зберігає інформацію про користувачів системи. До її основних полів належать id, email, password_hash, full_name, role та created_at. Поле id є первинним ключем. Email має бути унікальним, оскільки використовується для входу в систему. Поле password_hash зберігає хеш пароля, а не сам пароль у відкритому вигляді. Поле role визначає роль користувача: USER або ADMIN.

Таблиця cars є центральною таблицею системи. Вона зберігає інформацію про автомобільні оголошення. До її полів належать id, brand, model, year, price, currency, mileage_km, body_type, fuel_type, transmission, engine_volume_l, color, region, vin, description, source_platform, source_url, posted_at, fetched_at, updated_at та is_active. Така структура дозволяє зберігати як технічні характеристики автомобіля, так і службову інформацію про джерело оголошення та його актуальність.

Таблиця car_images використовується для зберігання зображень автомобілів. Вона містить id, car_id, url та position. Поле car_id є зовнішнім ключем, який посилається на таблицю cars. Завдяки цьому один автомобіль може мати декілька зображень, а поле position дозволяє визначити порядок їхнього відображення.

Таблиця favorites реалізує зв'язок між користувачами та автомобілями, доданими до обраного. Вона містить user_id, car_id та added_at. Оскільки один користувач може додати багато автомобілів, а один автомобіль може бути доданий до обраного багатьма користувачами, ця таблиця реалізує зв'язок багато-до-багатьох між users і cars. Первинним ключем може бути комбінація user_id і car_id.

Таблиця price_history зберігає історію зміни цін автомобіля. Вона містить id, car_id, price, currency та captured_at. Завдяки цій таблиці система може показувати користувачу, як змінювалася ціна оголошення з часом. Це також може використовуватися під час розрахунку AutoHub Deal Score.

Для підвищення швидкодії пошуку в базі даних доцільно використовувати індекси. У межах AutoHub індекси створюються для полів, які часто використовуються у фільтрах і пошуку: brand, model, price, year, source_platform, is_active, car_id у таблицях car_images, favorites і price_history. Це дозволяє прискорити виконання запитів до каталогу та аналітики.

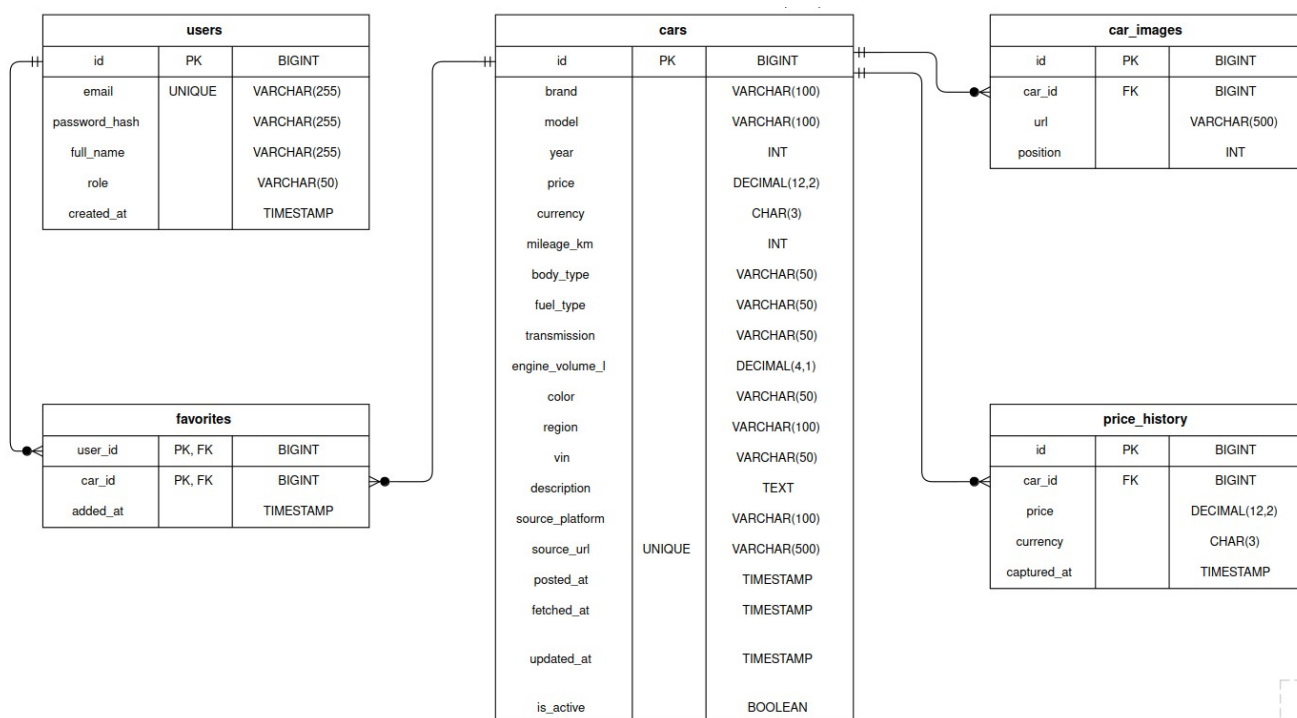


Рис. 4.3 ER-діаграма бази даних AutoHub

ER-діаграма бази даних AutoHub (див. рис. 4.3) відображає основні таблиці системи та зв'язки між ними. Центальною сутністю є cars, оскільки саме автомобільні оголошення є основним об'єктом роботи платформи. Таблиці users і favorites забезпечують персональну роботу користувачів з обраними автомобілями, car_images зберігає зображення авто, а price_history дозволяє відстежувати зміну ціни в часі.

4.4 Проєктування логіки AutoHub Deal Score

Однією з ключових особливостей AutoHub є механізм AutoHub Deal Score. Його призначення полягає у формуванні узагальненої оцінки вигідності автомобільного оголошення. На відміну від звичайного каталогу, де користувач бачить лише ціну, рік випуску та пробіг, Deal Score дозволяє отримати коротку аналітичну підказку про якість конкретної пропозиції.

Deal Score розраховується за 100-бальною шкалою. Чим вищий показник, тим більш привабливою вважається пропозиція. Оцінка не повинна замінювати повну перевірку автомобіля, але вона допомагає користувачу швидше відсіяти менш цікаві варіанти та зосередитися на потенційно вигідних оголошеннях.

Для розрахунку оцінки доцільно використати кілька груп факторів:

- ціновий фактор;
- фактор пробігу;
- фактор актуальності оголошення;
- фактор історії ціни;
- фактор популярності моделі.

Ціновий фактор є найважливішим, тому йому відводиться до 40 балів. Він визначає, наскільки ціна конкретного автомобіля відрізняється від середньої ціни аналогічних авто в системі. Якщо ціна нижча за середню для відповідної марки та моделі, автомобіль отримує більше балів. Якщо ціна вища за середній рівень, оцінка знижується.

Фактор пробігу оцінюється до 25 балів. Він враховує співвідношення пробігу з віком автомобіля. Наприклад, автомобіль старшого року випуску з дуже малим пробігом або новіший автомобіль із надмірним пробігом можуть потребувати додаткової перевірки. У межах системи цей фактор допомагає оцінити, наскільки пробіг виглядає адекватним для віку транспортного засобу.

Фактор актуальності оголошення оцінюється до 15 балів. Свіжі оголошення можуть бути більш актуальними, тоді як дуже старі можуть свідчити про низький попит, завищену ціну або втрату актуальності пропозиції. Для цього використовується дата публікації або дата завантаження оголошення в систему.

Фактор історії ціни оцінюється до 10 балів. Якщо для автомобіля зберігається історія зміни ціни, система може враховувати факт зниження ціни або стабільність вартості. Наприклад, якщо ціна знижувалася, це може свідчити про можливість вигіднішої покупки або готовність продавця до торгу.

Фактор популярності марки або моделі також оцінюється до 10 балів. Якщо певна марка або модель часто зустрічається в системі, це може свідчити про достатню кількість ринкових даних для порівняння. Крім того, популярні моделі зазвичай простіше порівнювати, обслуговувати та аналізувати з точки зору ринку.

Загальна структура оцінки може бути подана таким чином:

$\text{Deal Score} = \text{priceFactor} + \text{mileageFactor} + \text{freshnessFactor} + \text{priceHistoryFactor} + \text{popularityFactor}$ [1 - 4]

де:

- priceFactor — оцінка ціни відносно середньоринкової вартості, до 40 балів;
- mileageFactor — оцінка пробігу відносно віку автомобіля, до 25 балів;
- freshnessFactor — оцінка актуальності оголошення, до 15 балів;
- priceHistoryFactor — оцінка наявності та зміни історії ціни, до 10 балів;
- popularityFactor — оцінка популярності марки або моделі, до 10 балів.

Результат Deal Score може бути поданий користувачу у вигляді числового значення та текстової категорії.

Такий підхід робить аналітичну інформацію зрозумілою для користувача. Замість складних таблиць і числових показників він отримує короткий висновок, який допомагає швидше зорієнтуватися у каталозі.

4.5 Алгоритм розрахунку оцінки вигідності автомобіля

Алгоритм розрахунку AutoHub Deal Score починається з отримання даних про конкретний автомобіль. Система бере основні характеристики авто: марку, модель, рік випуску, ціну, пробіг, дату публікації, дату завантаження, історію зміни ціни та платформу-джерело. Після цього виконується пошук подібних автомобілів у базі даних.

Під подібними автомобілями розуміються оголошення з тією ж маркою та моделлю або максимально близькими параметрами. На основі цих даних система розраховує середню ринкову ціну. Якщо кількість подібних оголошень є достатньою, середнє значення використовується для оцінки цінового фактора. Якщо даних недостатньо, система може використовувати загальну статистику по марці або зменшувати вагу відповідного фактора.

Після цього система оцінює пробіг. Для цього визначається вік автомобіля та орієнтовний середній пробіг на рік. Якщо фактичний пробіг не перевищує очікуваний рівень, автомобіль отримує більшу кількість балів. Якщо пробіг значно вищий, оцінка зменшується.

Далі перевіряється актуальність оголошення. Якщо оголошення було опубліковане або оновлене нещодавно, воно отримує більше балів. Якщо оголошення є старим, оцінка знижується. Це дозволяє врахувати ймовірність того, що пропозиція вже неактуальна або має низький попит.

Наступним кроком є аналіз історії ціни. Якщо для автомобіля є записи в таблиці `price_history`, система може визначити, чи змінювалася ціна. Наявність історії ціни підвищує інформативність пропозиції, а зниження ціни може позитивно впливати на оцінку.

Після цього розраховується популярність марки або моделі. Для цього система може визначати кількість активних оголошень з такою самою маркою або моделлю. Якщо таких оголошень багато, система має більше даних для порівняння, а користувач отримує більш обґрунтовану оцінку.

Після розрахунку всіх факторів система підсумовує отримані бали. Якщо результат перевищує 100, він обмежується максимальним значенням. Якщо певний фактор не може бути розрахований через відсутність даних, система не повинна завершувати роботу з помилкою, а має використовувати нейтральне або зменшене значення. На рис. 4.4 представлена блок-схема алгоритму розрахунку AutoHub Deal Score

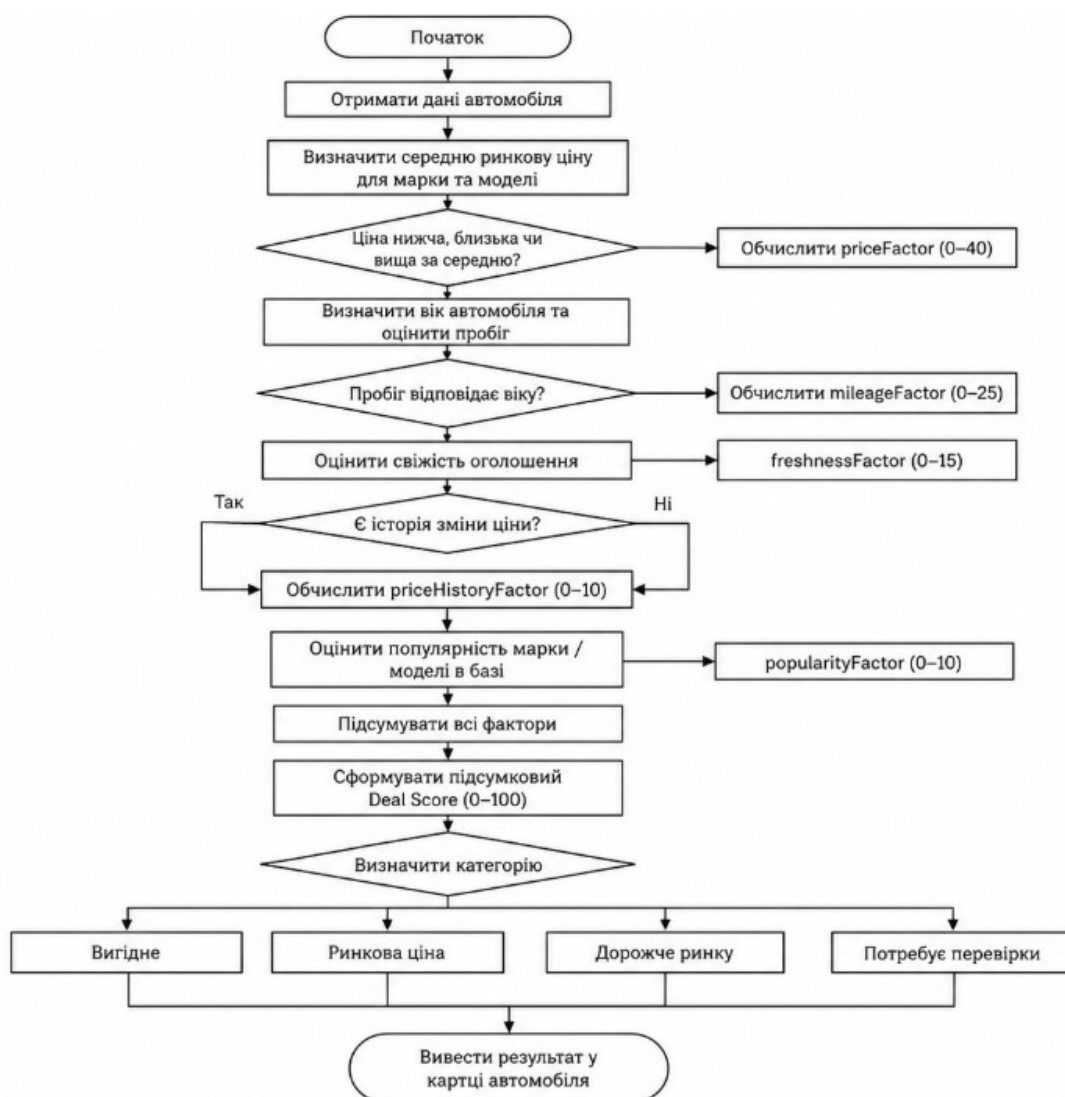


Рис. 4.4 Блок-схема алгоритму розрахунку AutoHub Deal Score

4.6 Діаграми послідовності основних сценаріїв

Діаграми послідовності використовуються для відображення взаємодії між користувачем, клієнтською частиною, сервером, сервісами та базою даних у межах конкретних сценаріїв. Для web-платформи AutoHub доцільно розглянути декілька основних сценаріїв: додавання автомобіля до обраного, розрахунок Deal Score та запуск оновлення даних адміністратором.

Перший сценарій — додавання автомобіля до обраного. Цей сценарій доступний лише авторизованому користувачу. Користувач натискає кнопку додавання до обраного на картці автомобіля. Frontend надсилає запит із JWT-токеном. Backend перевіряє токен, визначає користувача та передає запит до FavoriteService. Сервіс перевіряє існування автомобіля, створює запис у таблиці favorites і повертає підтвердження [21; 22].

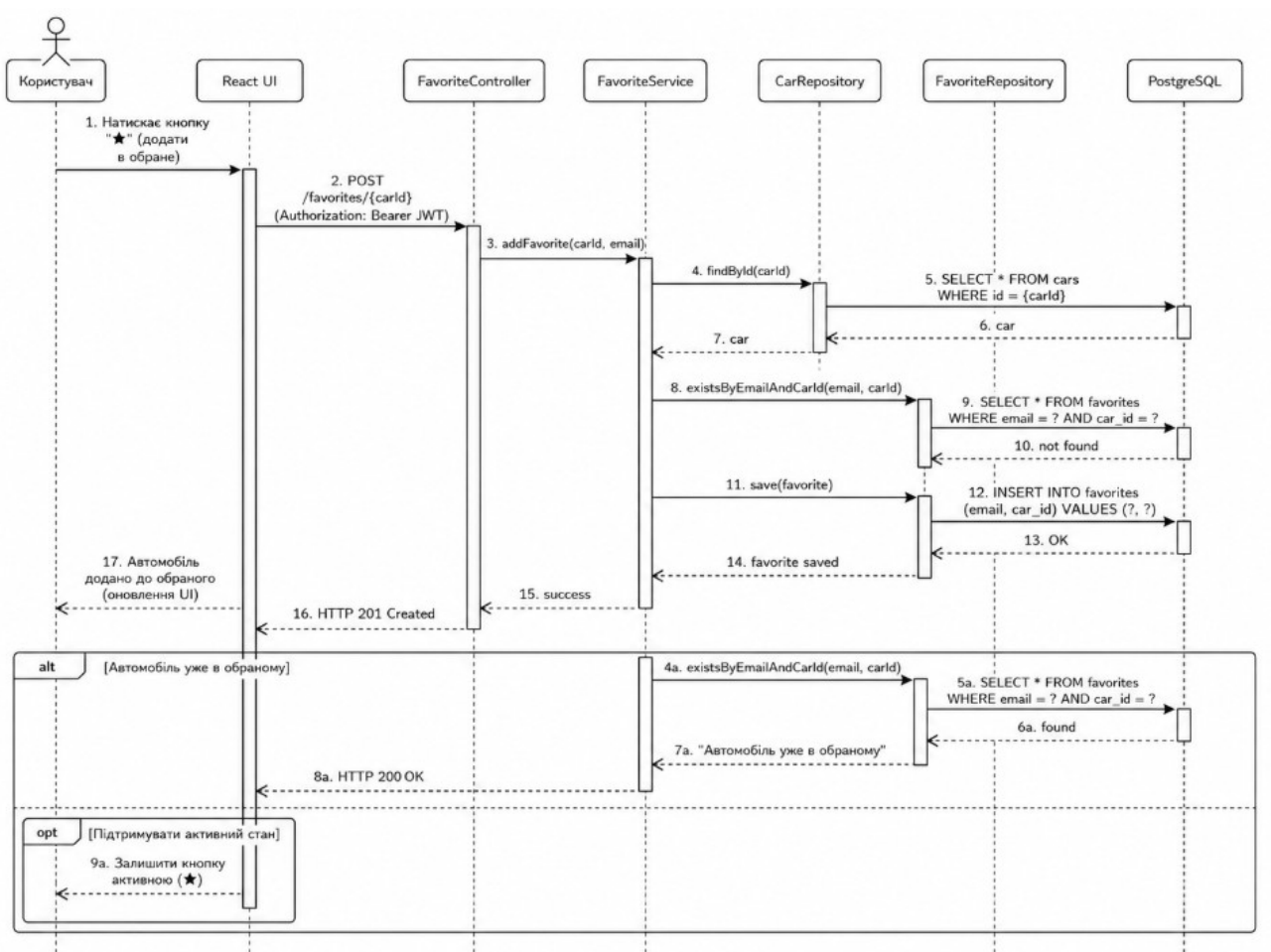


Рис. 4.5 Діаграма послідовності додавання автомобіля до обраного

Діаграма послідовності додавання автомобіля до обраного (див. рис. 4.5) відображає взаємодію користувача з клієнтською частиною, сервером, сервісом обраного та базою даних. Важливою частиною цього сценарію є перевірка авторизації, оскільки функція обраного належить до персональних можливостей користувача [7].

Другий сценарій — розрахунок AutoHub Deal Score. Він може виконуватися під час відкриття детальної картки автомобіля або під час формування списку каталогу. Backend отримує дані конкретного автомобіля, звертається до CarRepository для отримання схожих оголошень, до PriceHistoryRepository для аналізу історії цін, після чого DealScoreService розраховує загальну оцінку. Результат повертається у складі DTO та відображається у frontend-інтерфейсі [21; 22].

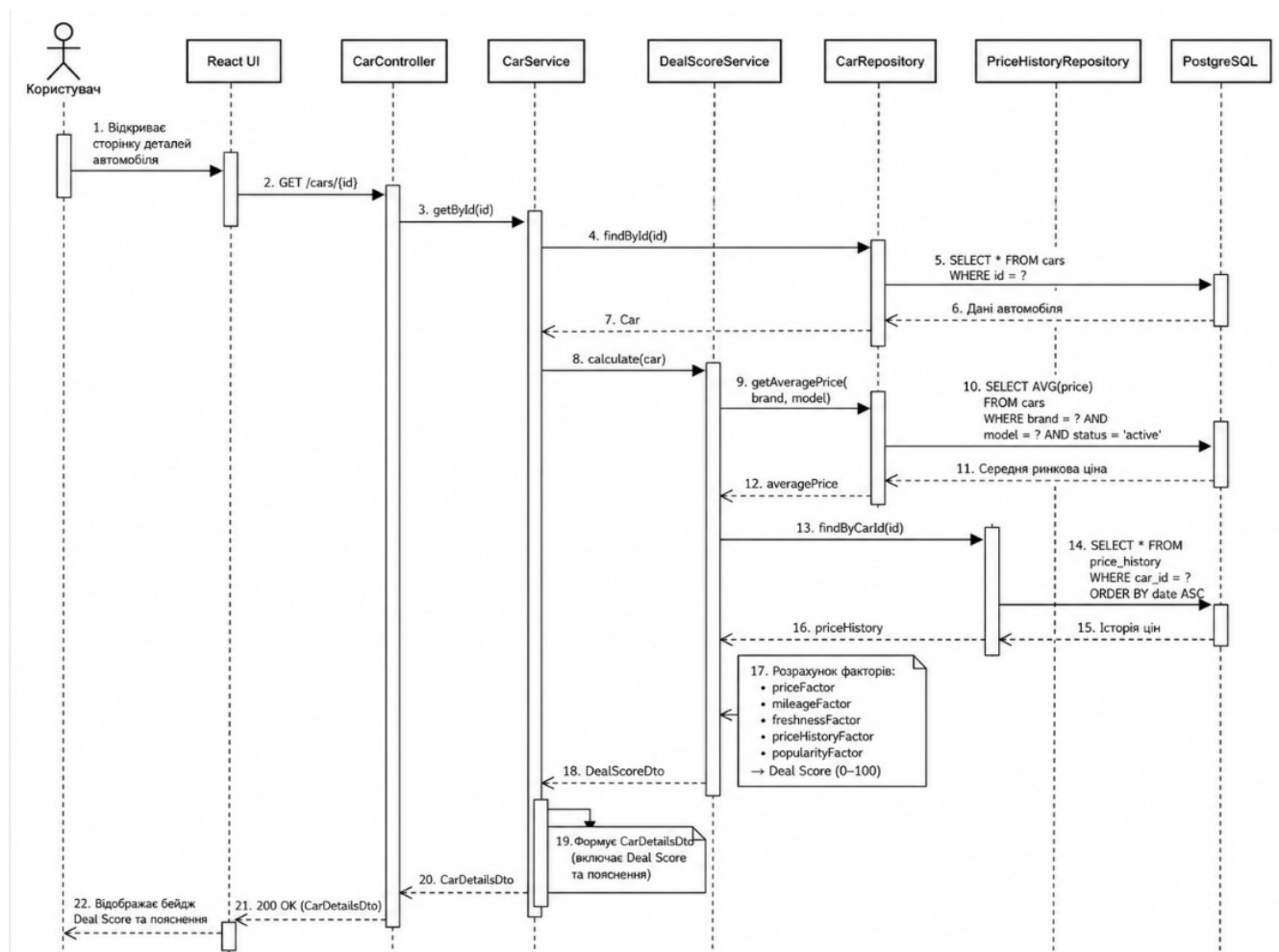


Рис. 4.6 Діаграма послідовності розрахунку AutoHub Deal Score

Діаграма послідовності розрахунку Deal Score (див. рис. 4.6) показує, як система формує оцінку вигідності автомобіля. У цьому сценарії важливою є взаємодія між сервісом автомобілів, репозиторіями та DealScoreService, який виконує розрахунок факторів і повертає підсумкову оцінку [14].

Третій сценарій — запуск оновлення даних адміністратором. Адміністратор відкриває адміністративну панель і натискає кнопку оновлення даних. Frontend надсилає захищений запит до backend. Сервер перевіряє роль користувача. Якщо користувач має роль ADMIN, запит передається до адміністративного сервісу або ScrapingService. Після завершення оновлення система повертає статус виконання [21; 22].

На рисунку 4.7 представлена діаграма послідовності запуску оновлення даних адміністратором

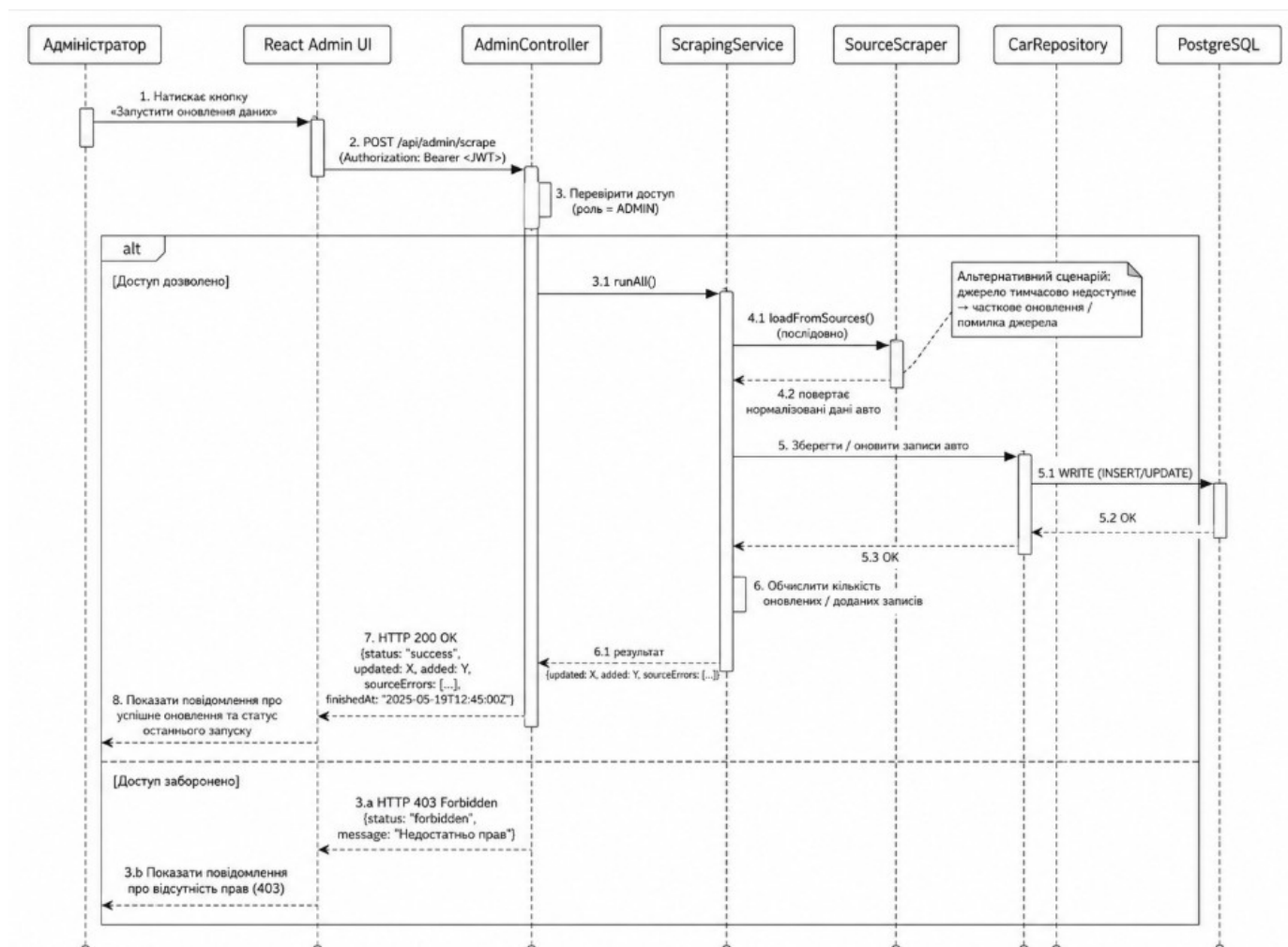


Рис. 4.7 Діаграма послідовності запуску оновлення даних адміністратором

5 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ПЛАТФОРМИ AUTOHUB

Програмна реалізація web-платформи AutoHub виконувалася відповідно до спроектованої архітектури, сформованих функціональних і нефункціональних вимог, а також визначеної структури бази даних. Основною метою реалізації було створення повноцінного клієнт-серверного web-застосунку, який забезпечує роботу з каталогом вживаних автомобілів, авторизацію користувачів, персональні функції, аналітичну обробку даних і механізм оцінювання вигідності пропозицій AutoHub Deal Score.

Під час реалізації було використано багатоваріантну структуру серверної частини. Такий підхід дозволяє розділити відповідальність між різними компонентами системи: контролерами, сервісами, репозиторіями, DTO-об'єктами, маперами, сутностями та конфігураційними класами. Завдяки цьому код є більш зрозумілим, підтримуваним і придатним до подальшого розширення [14].

Клієнтська частина реалізована як web-інтерфейс, через який користувач взаємодіє із системою. Вона відповідає за відображення сторінок, роботу з формами, надсилання запитів до backend, обробку відповідей сервера та візуальне представлення результатів. У межах frontend-частини реалізовано основні сторінки платформи: головну сторінку, каталог, сторінку деталей автомобіля, обране, порівняння, аналітику, профіль користувача, сторінки авторизації та адміністративну панель [21; 22].

5.1 Реалізація серверної частини

Серверна частина web-платформи AutoHub реалізована за допомогою мови програмування Java та фреймворку Spring Boot. Spring Boot використовується як основа backend-застосунку, оскільки він забезпечує швидке створення web-сервісів, автоматичну конфігурацію компонентів, підтримку REST API, інтеграцію з базою даних та засоби безпеки [7; 14; 19; 20].

Структура серверної частини побудована за принципом розділення на логічні пакети. Основними пакетами є:

- `controller` — класи, які приймають HTTP-запити від клієнтської частини;
- `service` — класи, що реалізують бізнес-логіку системи;
- `repository` — інтерфейси доступу до бази даних;
- `entity` — класи, що відповідають таблицям бази даних;
- `dto` — об'єкти передавання даних між `frontend` і `backend`;
- `mapper` — класи для перетворення сутностей у DTO;
- `security` — класи, пов'язані з авторизацією, JWT і перевіркою користувачів;
- `config` — конфігураційні класи системи;
- `exception` — класи для обробки помилок.

Контролери реалізують зовнішній REST API системи. Наприклад, `CarController` відповідає за роботу з автомобілями. Він містить методи для отримання списку авто, перегляду детальної інформації та порівняння декількох автомобілів. При зверненні користувача до каталогу `frontend` надсилає запит до відповідного `endpoint`, після чого контролер передає параметри до сервісного шару [21; 22].

`FavoriteController` реалізує роботу з обраними автомобілями. Він дозволяє отримати список обраних авто користувача, додати автомобіль до обраного або видалити його. Ці дії доступні лише авторизованим користувачам, тому під час обробки запиту враховується email або дані поточного користувача, отримані з контексту безпеки.

`AnalyticsController` відповідає за аналітичні запити. Через нього `frontend` може отримувати загальну статистику ринку, дані про середні ціни, кількість пропозицій, популярні марки та інші показники. Це дозволяє відокремити аналітичну логіку від основної логіки каталогу [21; 22].

Сервісний шар є центральною частиною backend-логіки. Наприклад, `CarService` реалізує основні операції з автомобілями: отримання списку, фільтрацію, пошук за параметрами, отримання автомобіля за ідентифікатором та порівняння декількох авто. Саме в цьому шарі формується логіка роботи з фільтрами та викликаються репозиторії [14].

Для реалізації пошуку й фільтрації використовується підхід на основі специфікацій. Клас `CarSpecification` формує умови пошуку залежно від параметрів, які передає користувач. Якщо користувач задає марку, модель, рік, ціну, пробіг, тип кузова, пальне, трансмісію або регіон, ці параметри перетворюються у відповідні умови запиту до бази даних. Такий підхід дозволяє гнучко комбінувати фільтри та не створювати окремі методи репозиторію для кожної можливої комбінації параметрів [14].

`FavoriteService` відповідає за бізнес-логіку обраного. Перед додаванням автомобіля до обраного сервіс перевіряє, чи існує користувач, чи існує автомобіль, а також чи не було вже додано це авто раніше. Це дозволяє уникнути дублювання записів і забезпечити коректність персональних списків користувачів.

`AnalyticsService` виконує формування аналітичних показників. Він може використовувати дані з таблиць `cars` і `price_history` для розрахунку середніх цін, кількості пропозицій, розподілу автомобілів за брендами, роками випуску або іншими параметрами. Аналітичний модуль важливий для того, щоб користувач міг бачити не лише окремі оголошення, а й загальну ситуацію на ринку.

`DealScoreService` реалізує механізм `AutoHub Deal Score`. Він отримує дані про конкретний автомобіль, аналізує його ціну, пробіг, актуальність оголошення, історію ціни та популярність моделі. Після цього сервіс формує підсумкову оцінку за 100-бальною шкалою та повертає її у вигляді DTO. Такий підхід дозволяє розміщувати оцінку як у картці автомобіля, так і на сторінці деталей [1 - 4].

Для роботи з базою даних використовуються JPA-репозиторії. `CarRepository` відповідає за роботу з таблицею автомобілів, `UserRepository` — з користувачами, `FavoriteRepository` — з обраними авто, `PriceHistoryRepository` — з історією зміни цін. Використання Spring Data JPA дозволяє скоротити обсяг коду для стандартних операцій, таких як пошук, збереження, оновлення та видалення записів [7; 14; 19; 20].

Для передачі даних між backend і frontend використовуються DTO-об'єкти. Наприклад, `CarDto` містить скорочену інформацію для відображення автомобіля у каталозі, а `CarDetailsDto` містить розширену інформацію для сторінки деталей. Такий підхід дозволяє не передавати на клієнтську частину зайві службові поля та контролювати структуру відповіді API [21; 22].

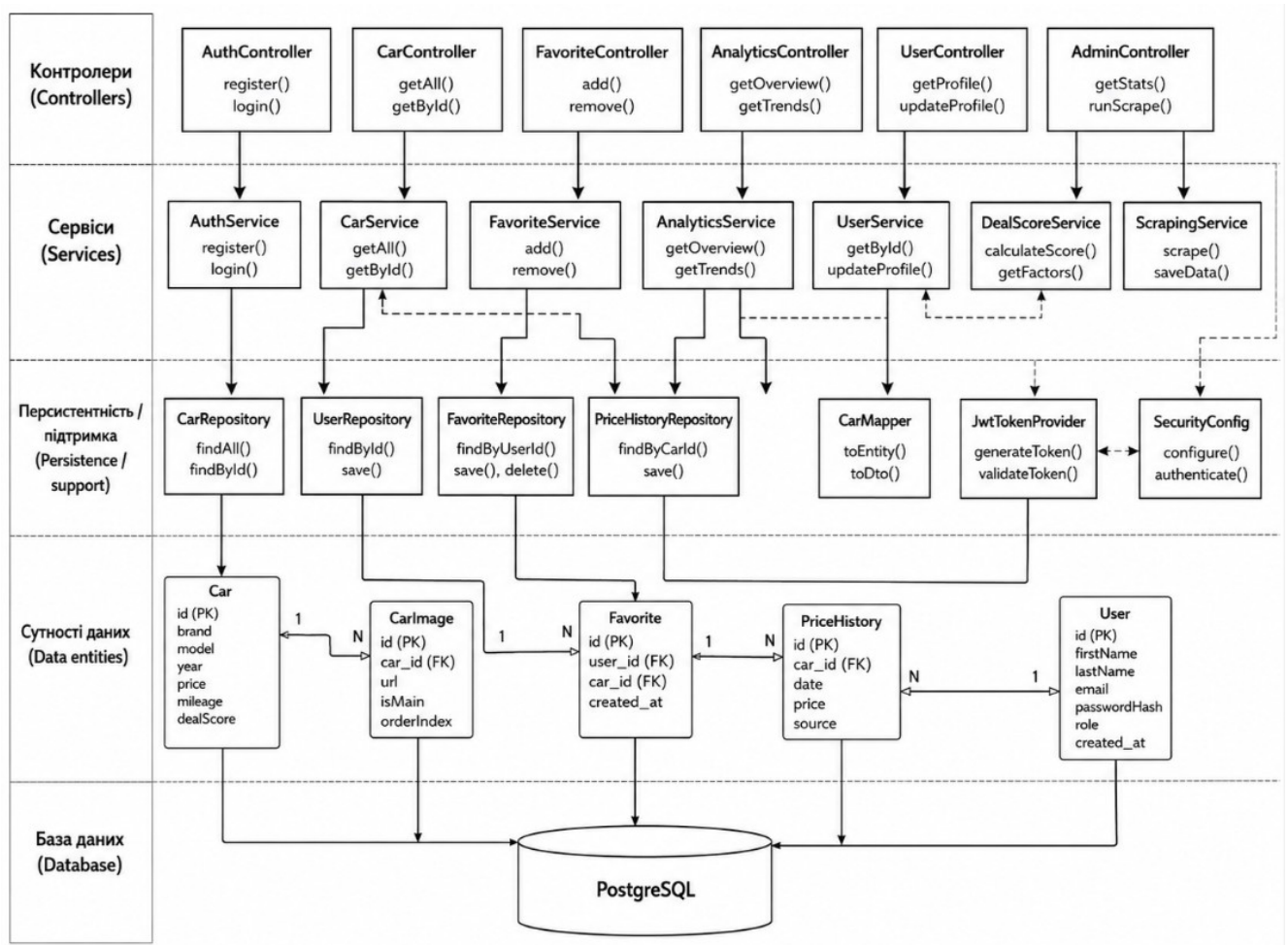


Рис. 5.1 Структура серверної частини web-платформи AutoHub

Структура серверної частини (див. рис. 5.1) відображає взаємодію між контролерами, сервісами, репозиторіями, DTO-об'єктами та сутностями. Кожен шар виконує окрему роль: контролери приймають запити, сервіси реалізують бізнес-логіку, репозиторії працюють із базою даних, а DTO забезпечують передачу даних у зручному для frontend форматі.

5.2 Реалізація клієнтської частини

Клієнтська частина web-платформи AutoHub реалізована за допомогою React. React дозволяє створювати інтерфейс на основі компонентів, що є зручним для розробки web-застосунків із великою кількістю сторінок і повторюваних елементів. У межах AutoHub компонентний підхід використовується для реалізації карток автомобілів, форм пошуку, фільтрів, таблиць порівняння, блоків аналітики, навігації та форм авторизації [21; 22].

Для запуску та збирання frontend-частини використовується Vite. Він забезпечує швидке оновлення інтерфейсу під час розробки та формування оптимізованої production-збірки. Це дозволяє прискорити процес розробки та перевірки web-інтерфейсу.

Клієнтська частина складається зі сторінок і компонентів. До основних сторінок належать:

- головна сторінка;
- сторінка каталогу автомобілів;
- сторінка деталей автомобіля;
- сторінка обраних автомобілів;
- сторінка порівняння;
- сторінка аналітики;
- сторінка профілю користувача;
- сторінка входу;
- сторінка реєстрації;
- адміністративна панель.

Головна сторінка виконує роль початкової точки входу до системи. Вона повинна швидко пояснювати користувачу призначення платформи та надавати перехід до каталогу автомобілів. Основний акцент робиться не на рекламному описі, а на швидкому доступі до функцій пошуку та перегляду пропозицій.

Сторінка каталогу є однією з ключових частин системи. Вона відображає список автомобілів у вигляді карток або таблиці. Кожна картка містить основну інформацію: марку, модель, рік випуску, ціну, пробіг, регіон, платформу-джерело та коротку оцінку пропозиції. Також на картці можуть бути розміщені кнопки для додавання до обраного, відкриття деталей і додавання до порівняння.

Фільтрація в каталозі дозволяє користувачу швидко звузити список оголошень. Для цього передбачаються фільтри за маркою, моделлю, ціною, роком випуску, пробігом, типом кузова, паливом, трансмісією, регіоном і джерелом оголошення. Після зміни параметрів frontend надсилає запит до backend, а отримані результати відображаються користувачу.

Сторінка деталей автомобіля містить повнішу інформацію про конкретне авто. На ній відображаються фото, технічні характеристики, опис, ціна, пробіг, дата публікації, посилання на джерело, історія зміни ціни та AutoHub Deal Score. Ця сторінка є важливою для прийняття рішення, оскільки користувач отримує не тільки окремі характеристики, а й аналітичну оцінку пропозиції.

Сторінка обраного призначена для збереження автомобілів, які зацікавили користувача. Вона дозволяє швидко повернутися до потрібних пропозицій без повторного пошуку. Це особливо важливо для користувачів, які порівнюють декілька варіантів протягом кількох днів або очікують зміни ціни.

Сторінка порівняння дозволяє зіставити декілька автомобілів за ключовими параметрами. Інформація подається у табличному вигляді, щоб користувач міг швидко побачити різницю між моделями, цінами, роками випуску, пробігом, типом пального, трансмісією та іншими характеристиками. Порівняння зменшує кількість ручних дій і робить вибір більш структурованим.

Сторінка аналітики відображає узагальнені показники ринку. На ній можуть бути представлені кількість оголошень, середня ціна, популярні марки, розподіл

за роками випуску, середні ціни за брендами або моделями. Такий розділ дозволяє користувачу краще розуміти загальну ситуацію на ринку.

Профіль користувача містить персональну інформацію: ім'я, email, роль, дату реєстрації та коротку статистику дій. Також у профілі може бути передбачене редагування імені або зміна пароля. Ця сторінка робить систему більш персоналізованою та завершеною з точки зору користувацького сценарію.

Адміністративна панель доступна лише користувачам із роллю ADMIN. Вона використовується для перегляду статистики системи, кількості користувачів, автомобілів, записів історії цін, обраних авто та запуску оновлення даних. Наявність адміністративної панелі дозволяє контролювати стан системи та демонструє розмежування прав доступу.

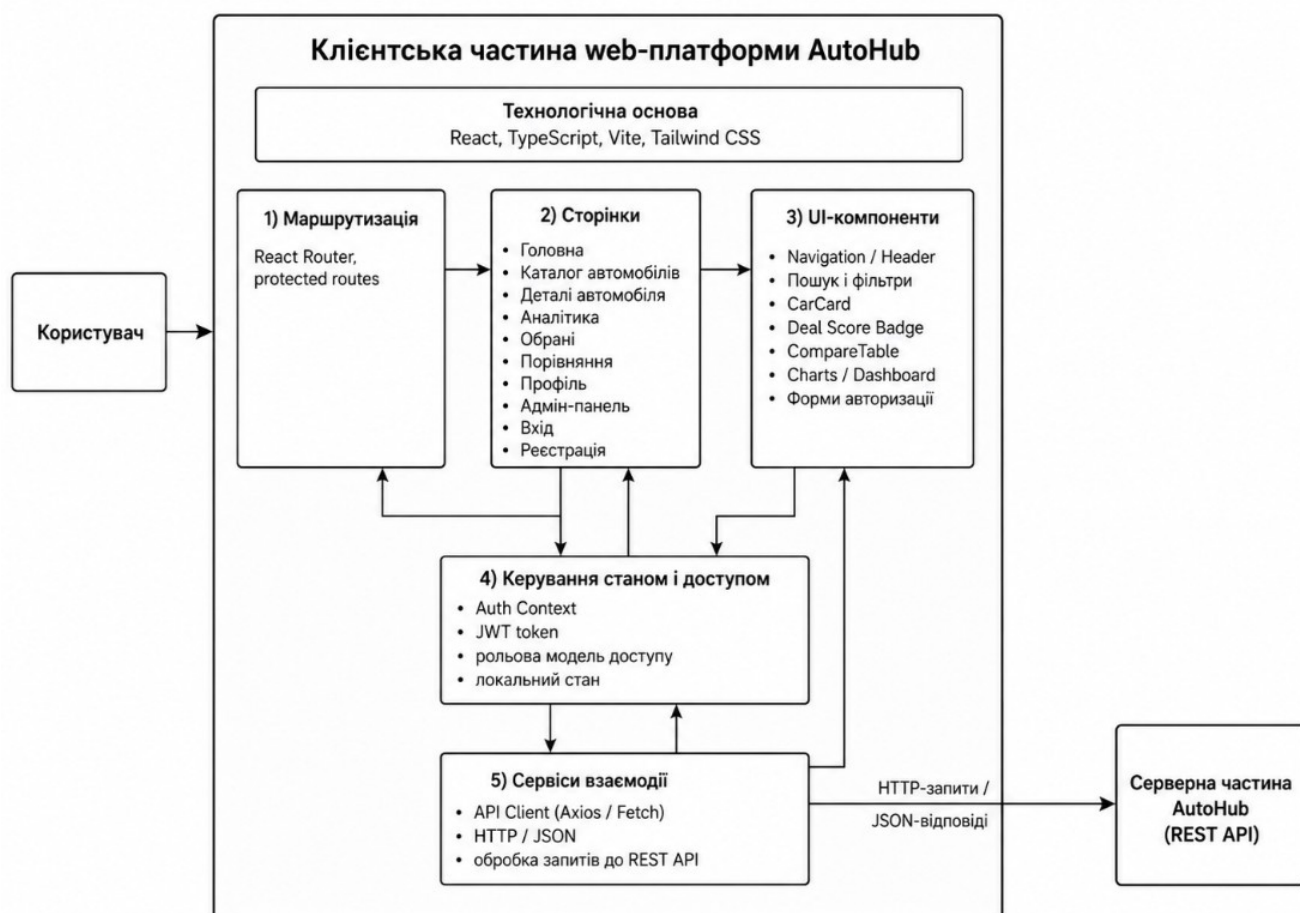


Рис. 5.2 Структура клієнтської частини web-платформи AutoHub

Структура клієнтської частини (див. рис. 5.2) відображає основні сторінки та компоненти web-застосунку. Взаємодія користувача з платформою відбувається через інтерфейс, який надсилає запити до backend і відображає отримані дані у вигляді каталогу, карток, таблиць, графіків та форм.

5.3 Реалізація авторизації та розмежування ролей

Авторизація та розмежування ролей є важливою частиною web-платформи AutoHub, оскільки система містить як публічні, так і захищені розділи. Публічна частина доступна гостям, тоді як персональні функції та адміністративні можливості повинні бути доступні лише після входу в систему [7].

Для реалізації безпеки використовується Spring Security. Система підтримує реєстрацію та вхід користувача. Під час реєстрації користувач передає email, пароль і повне ім'я. Перед збереженням пароль хешується за допомогою BCrypt, після чого у базі даних зберігається не відкритий пароль, а його захищений хеш [19; 20].

Під час входу користувач передає email і пароль. Сервер перевіряє наявність користувача, порівнює пароль із хешем і у випадку успішної перевірки формує JWT-токен. Цей токен повертається клієнтській частині та використовується для подальших запитів до захищених endpoint.

JWT-токен містить інформацію, необхідну для ідентифікації користувача. При кожному запиті до захищеного ресурсу frontend додає токен у заголовок запиту. На backend-стороні JwtAuthenticationFilter перевіряє токен, визначає користувача та встановлює автентифікацію у контексті Spring Security [19 - 25].

Розмежування доступу здійснюється за ролями. У системі використовуються ролі USER і ADMIN. Звичайний користувач має доступ до персональних функцій: обраного, профілю, порівняння та інших дій, пов'язаних із вибором автомобіля. Адміністратор додатково має доступ до адміністративної панелі та службових функцій.

На frontend-стороні також реалізується умовне відображення елементів меню. Наприклад, пункт «Адмін» повинен бути видимим лише для користувача з роллю ADMIN. Якщо звичайний користувач намагається перейти на адміністративну сторінку вручну, система повинна або показати повідомлення про заборону доступу, або перенаправити його на іншу сторінку [21; 22].

Така реалізація дозволяє забезпечити базовий рівень безпеки та логічне розділення сценаріїв роботи різних користувачів. Публічні функції залишаються доступними без входу, персональні функції потребують авторизації, а адміністративні функції додатково перевіряють роль користувача.

5.4 Реалізація каталогу, фільтрації та порівняння автомобілів

Каталог автомобілів є центральним функціональним модулем AutoHub. Саме через каталог користувач починає основну взаємодію з платформою. У каталозі відображаються автомобілі, які зберігаються в базі даних, а також надаються інструменти для пошуку, сортування та фільтрації [1 - 6].

На backend-стороні отримання автомобілів реалізується через `CarController` і `CarService`. Коли користувач відкриває каталог або змінює параметри фільтрації, frontend надсилає GET-запит із відповідними параметрами. Сервер приймає ці параметри, формує об'єкт фільтра та передає його до сервісу [21; 22].

У сервісному шарі використовується `CarSpecification`, який дозволяє динамічно створювати умови пошуку. Наприклад, якщо користувач задає тільки марку і ціну, запит буде містити лише ці умови. Якщо користувач додатково вказує рік, пробіг, тип пального або трансмісію, система додає відповідні умови до запиту.

Такий підхід є зручним, оскільки кількість можливих комбінацій фільтрів є великою. Без використання специфікацій довелося б створювати багато окремих методів для різних варіантів пошуку. Використання динамічних умов дозволяє зробити фільтрацію більш гнучкою та масштабованою.

Для зручності користувача каталог підтримує пагінацію та сортування. Пагінація дозволяє не завантажувати всі записи одразу, а відображати їх частинами. Це покращує швидкодію системи та зменшує навантаження на frontend. Сортування може виконуватися за ціною, роком випуску, пробігом, датою публікації або іншими параметрами [21; 22].

Порівняння автомобілів реалізується як окремий сценарій. Користувач обирає декілька авто, після чого система отримує їхні дані та відображає у вигляді таблиці. Порівняння дозволяє зіставити характеристики: марку, модель, рік, ціну, пробіг, тип кузова, паливе, трансмісію, регіон, джерело та Deal Score [1 - 4].

На backend-стороні порівняння може бути реалізоване через метод `compareByIds`, який приймає список ідентифікаторів автомобілів. Система перевіряє, чи передано допустиму кількість авто, знаходить записи в базі даних і перетворює їх у DTO для відображення на клієнтській частині.

Рис. 5.3 Інтерфейс головної сторінки AutoHub

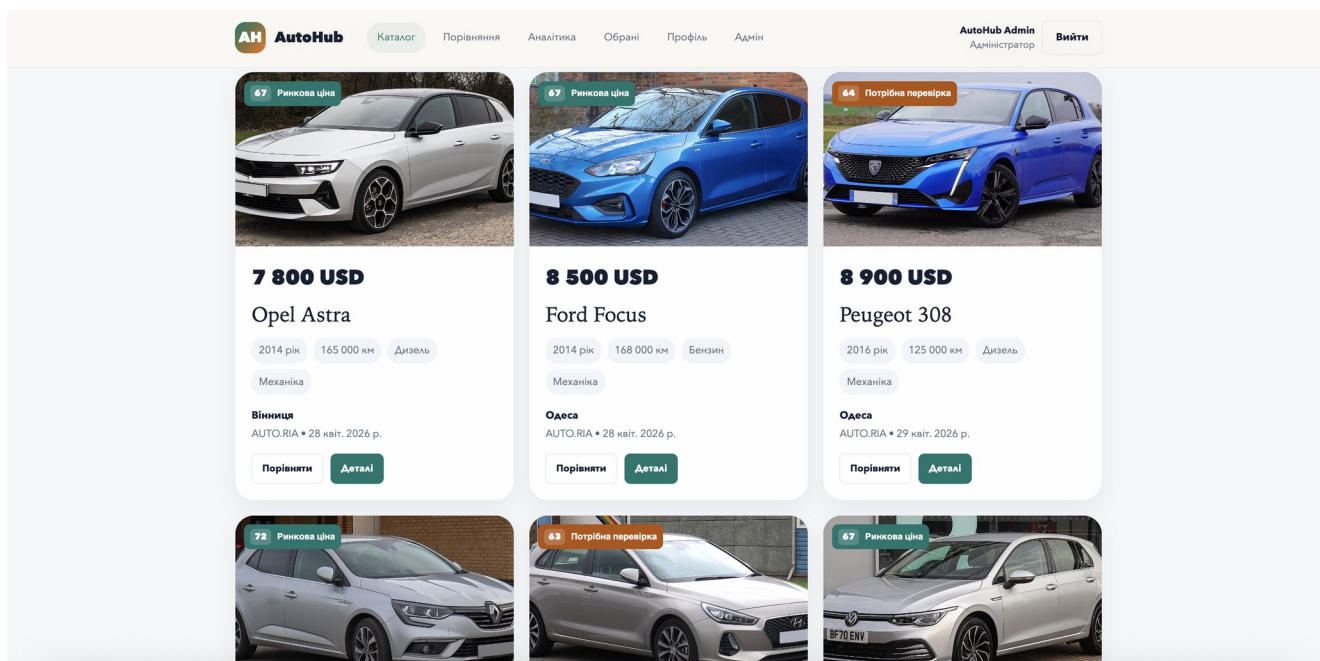


Рис. 5.4 Інтерфейс каталогу автомобілів

Сторінка каталогу (див. рис. 5.4) дозволяє користувачу працювати з великим обсягом автомобільних оголошень. Завдяки фільтрації, сортуванню, пагінації та переходу до детальної картки користувач може швидко знайти релевантні пропозиції. Додаткові дії, такі як обране та порівняння, роблять процес вибору автомобіля більш зручним і структурованим.

5.5 Реалізація профілю користувача

Профіль користувача є персональним розділом web-платформи AutoHub. Його призначення полягає у відображенні основних даних про користувача та наданні доступу до персональної інформації, пов'язаної з його активністю в системі.

У профілі можуть відображатися такі дані:

- повне ім'я користувача;
- email;
- роль у системі;
- дата реєстрації;

- кількість обраних автомобілів;
- коротка статистика персональної активності;
- середня ціна автомобілів в обраному;
- найбільш часта марка серед збережених авто.

На backend-стороні профіль реалізується через `UserController` і `UserService`. Endpoint `GET /api/users/me` дозволяє отримати дані поточного користувача. Для цього backend визначає користувача за JWT-токеном, знаходить його у базі даних і повертає DTO з потрібною інформацією.

Для редагування профілю може використовуватися endpoint `PATCH /api/users/me`. Користувач має змогу змінити повне ім'я. Email доцільно залишити незмінним або змінювати лише з додатковою перевіркою унікальності, оскільки він використовується як ідентифікатор для авторизації.

Зміна пароля може бути винесена в окремий endpoint. У такому сценарії користувач передає старий пароль і новий пароль. Backend перевіряє правильність старого пароля, хешує новий і зберігає його в базі даних. Такий підхід забезпечує додатковий рівень безпеки [7].

На frontend-стороні профіль має бути реалізований як спокійна утилітарна сторінка. Вона не повинна виглядати як рекламний екран, оскільки її основна функція — надати користувачу доступ до особистих даних і короткої персональної статистики [21; 22].

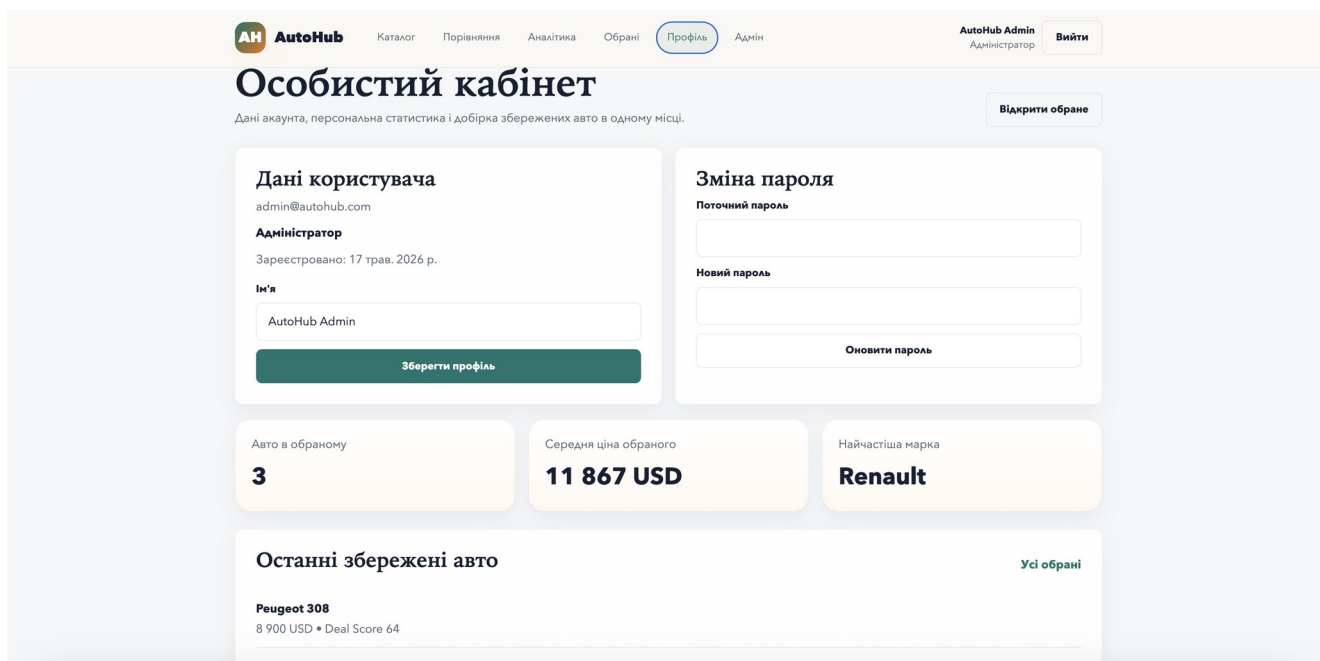


Рис. 5.5 Інтерфейс профілю користувача

Профіль користувача (див. рис. 5.5) доповнює систему персональним сценарієм. Завдяки йому користувач може бачити власні дані, контролювати обрані авто та отримувати короткий огляд своєї активності на платформі.

5.6 Реалізація адміністративної панелі

Адміністративна панель призначена для користувачів із роллю ADMIN. Вона забезпечує службові можливості керування системою та перегляду її стану. Наявність адміністративної панелі робить web-платформу більш наближеною до реального програмного продукту, оскільки дозволяє не лише користуватися системою, а й контролювати її наповнення.

Основні функції адміністративної панелі:

- перегляд кількості автомобілів у базі;
- перегляд кількості користувачів;
- перегляд кількості обраних авто;
- перегляд кількості записів історії цін;
- запуск оновлення даних;

- перегляд останніх доданих автомобілів;
- перегляд користувачів;
- можливість керування активністю оголошень.

На backend-стороні адміністративні функції реалізуються через окремий контролер, наприклад `AdminController`. Для отримання статистики може використовуватися endpoint `GET /api/admin/stats`. Він звертається до відповідних репозиторіїв, підраховує кількість записів і повертає дані на frontend [21; 22].

Для запуску оновлення даних може використовуватися endpoint `POST /api/admin/scrape` або аналогічний службовий endpoint. Він викликає сервіс оновлення даних, який запускає збір або оновлення автомобільних оголошень. У демонстраційній версії цей процес може працювати з demo-даними, але архітектурно він відокремлений як службова функція адміністратора.

Доступ до адміністративної панелі обмежується роллю ADMIN. Перевірка має виконуватися як на frontend, так і на backend. Frontend приховує пункт меню для звичайного користувача, а backend остаточно перевіряє права доступу при кожному запиті до адміністративних endpoint. Це важливо, оскільки приховування кнопки в інтерфейсі не є достатнім захистом [21; 22].

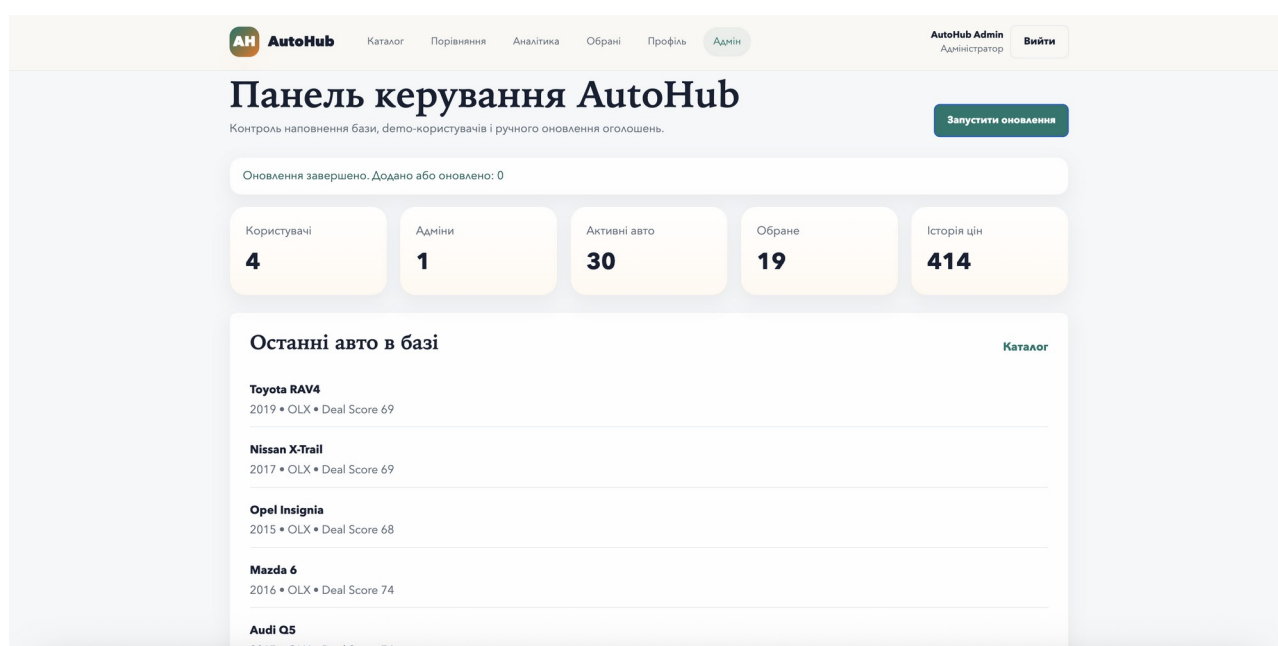


Рис. 5.6 Інтерфейс адміністративної панелі

Адміністративна панель (див. рис. 5.6) дозволяє контролювати основні показники системи та виконувати службові дії. Завдяки їй платформа має не лише користувацьку, а й керуючу частину, що є важливим для підтримки актуальності даних.

5.7 Реалізація AutoHub Deal Score

AutoHub Deal Score є унікальною функціональною особливістю платформи. Його реалізація спрямована на те, щоб надати користувачу коротку і зрозумілу оцінку вигідності конкретного автомобільного оголошення. На відміну від звичайного перегляду ціни або пробігу, Deal Score формує узагальнений показник, який враховує кілька факторів одночасно.

На backend-стороні механізм реалізується в окремому сервісі DealScoreService. Це дозволяє відокремити логіку оцінювання від основної логіки каталогу або деталей автомобіля. Такий підхід є правильним з точки зору архітектури, оскільки оцінка вигідності є окремою бізнес-функцією.

Для розрахунку Deal Score використовуються такі фактори:

- ціновий фактор;
- фактор пробігу;
- фактор актуальності оголошення;
- фактор історії зміни ціни;
- фактор популярності марки або моделі.

Ціновий фактор визначає, наскільки ціна конкретного автомобіля відрізняється від середньої ціни подібних авто. Для цього система знаходить оголошення з такою самою маркою та моделлю або близькими характеристиками. Якщо ціна нижча за середню, автомобіль отримує більше балів. Якщо ціна вища, оцінка знижується.

Фактор пробігу враховує співвідношення пробігу та року випуску. Якщо пробіг виглядає адекватним для віку автомобіля, оцінка буде вищою. Якщо пробіг

надмірний, система зменшує кількість балів. Це дозволяє уникнути ситуації, коли низька ціна автоматично сприймається як вигідна, хоча автомобіль може мати значне зношення.

Фактор актуальності оголошення враховує дату публікації або завантаження. Свіжі оголошення оцінюються вище, тоді як старі можуть потребувати додаткової перевірки. Це пов'язано з тим, що довге перебування оголошення на ринку може свідчити про завищену ціну або низьку привабливість пропозиції.

Фактор історії ціни враховує наявність записів у таблиці `price_history`. Якщо система має інформацію про зміну ціни, вона може оцінити, чи знижувалася ціна з часом. Наявність такої історії робить пропозицію більш прозорою для користувача.

Фактор популярності марки або моделі дозволяє врахувати кількість подібних оголошень у системі. Якщо марка або модель часто зустрічається, система має більше даних для порівняння. Це підвищує якість аналітичної оцінки.

Результат Deal Score повертається у вигляді DTO, наприклад `DealScoreDto`. Він може містити: [1; 2; 3; 4]

- загальну кількість балів;
- текстову категорію;
- короткі пояснення факторів;
- відсоток відхилення ціни від середньої;
- ознаку відповідності пробігу віку автомобіля;
- інформацію про історію ціни.

На frontend-стороні Deal Score відображається у вигляді бейджа або окремого блоку на сторінці деталей автомобіля (див. рис. 5.7). Наприклад, користувач може бачити статус «Вигідна пропозиція», «Ринкова ціна», «Дорожче ринку» або «Потребує перевірки». Також можуть відображатися короткі пояснення: «ціна нижча за середню», «пробіг відповідає віку», «є історія зміни ціни» [21; 22].

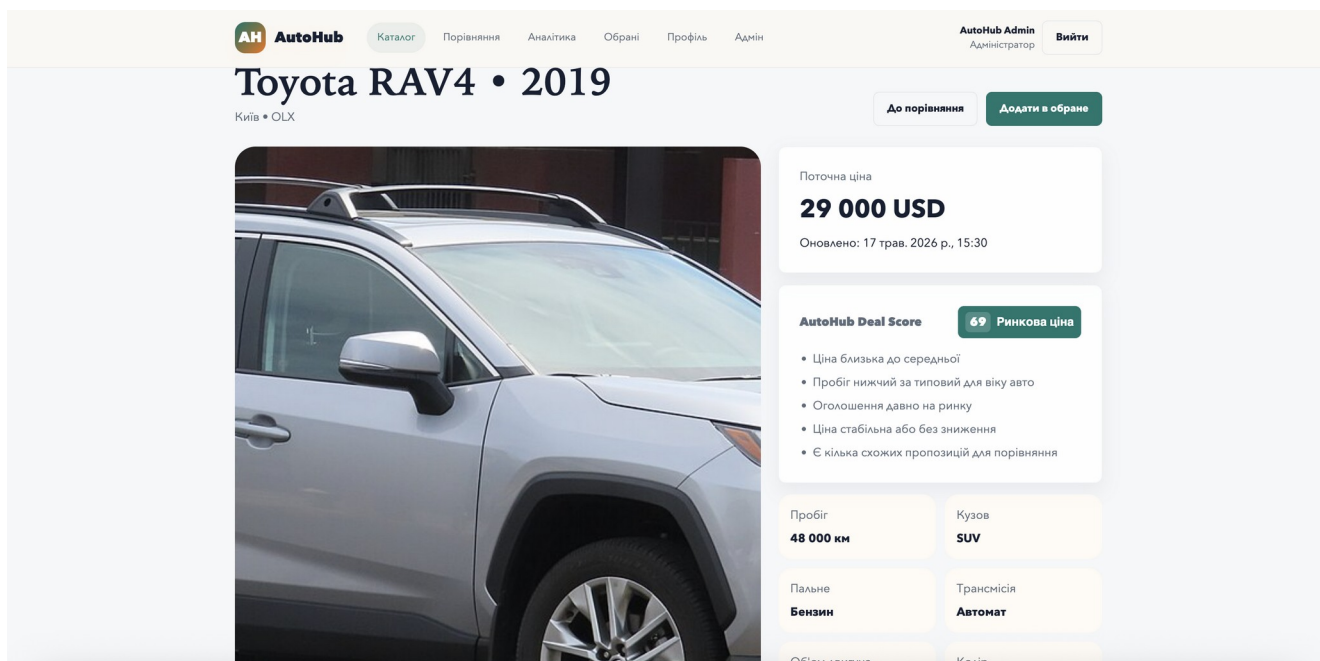


Рис. 5.7 Інтерфейс сторінки деталей автомобіля з AutoHub Deal Score

Реалізація Deal Score підвищує практичну цінність платформи AutoHub. Завдяки цьому система не просто показує автомобілі, а допомагає користувачу інтерпретувати дані та швидше приймати рішення. Саме ця функція є основною відмінністю AutoHub від звичайних сервісів оголошень.

6 ТЕСТУВАННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ РОБОТИ СИСТЕМИ

Тестування web-платформи AutoHub виконувалося з метою перевірки коректності роботи основних функцій системи, відповідності реалізованого програмного забезпечення сформованим вимогам, а також оцінки зручності взаємодії користувача з платформою. Оскільки AutoHub є клієнт-серверним web-застосунком, тестування охоплює як серверну частину, так і клієнтський інтерфейс [1 - 4].

Основна увага під час перевірки приділялася таким сценаріям: реєстрація та авторизація користувача, робота з каталогом автомобілів, пошук і фільтрація, перегляд детальної картки авто, додавання автомобіля до обраного, порівняння декількох пропозицій, перегляд аналітики, робота профілю користувача, доступ до адміністративної панелі та коректність розрахунку AutoHub Deal Score [7].

Тестування проводилося у локальному середовищі з використанням PostgreSQL, backend-частини на Spring Boot та frontend-частини на React. Для перевірки роботи API використовувалися web-інтерфейс застосунку, Swagger/OpenAPI та тестові HTTP-запити. Для модульної перевірки окремих backend-компонентів можуть використовуватися JUnit і Mockito, що дозволяють перевірити логіку сервісів незалежно від повного запуску системи.

6.1 План тестування

План тестування було сформовано відповідно до функціональних вимог, визначених у попередніх розділах. Його основна мета — перевірити, чи реалізовані ключові сценарії роботи користувача та адміністратора, а також чи відповідає система базовим вимогам до швидкодії, безпеки та зручності використання.

Тестування web-платформи AutoHub передбачає перевірку таких груп функцій:

- авторизація та реєстрація користувачів;
- розмежування доступу за ролями USER і ADMIN;
- перегляд каталогу автомобілів;
- пошук і фільтрація оголошень;
- перегляд детальної картки автомобіля;
- додавання автомобіля до обраного;
- видалення автомобіля з обраного;
- порівняння декількох автомобілів;
- перегляд аналітичних даних;
- відображення історії зміни ціни;
- розрахунок AutoHub Deal Score;
- робота профілю користувача;
- робота адміністративної панелі;
- перевірка обробки помилок.

Для перевірки було визначено декілька типових ролей користувачів. Гість використовується для перевірки публічної частини системи. Зареєстрований користувач використовується для перевірки персональних функцій: обраного, порівняння, профілю та дій із каталогом. Адміністратор використовується для перевірки службових функцій, зокрема адміністративної панелі, статистики та запуску оновлення даних [1 - 6].

Тестування проводиться за принципом проходження реальних користувацьких сценаріїв. Наприклад, спочатку перевіряється відкриття головної сторінки й каталогу без авторизації, потім реєстрація нового користувача, вхід у систему, додавання авто до обраного, порівняння автомобілів і перегляд аналітики. Окремо перевіряється сценарій входу адміністратора та доступ до адміністративної панелі [7].

Важливим етапом є перевірка негативних сценаріїв. До них належать: спроба входу з неправильним паролем, спроба доступу до профілю без авторизації, спроба звичайного користувача перейти до адміністративної панелі, передавання некоректних параметрів фільтрації або запит неіснуючого

автомобіля. Такі перевірки дозволяють оцінити, наскільки система коректно реагує на помилки.

6.2 Функціональне тестування

Функціональне тестування спрямоване на перевірку того, чи виконує система всі основні функції, передбачені вимогами. Для цього було сформовано набір тест-кейсів, які охоплюють основні сценарії взаємодії користувача з платформою.

Таблиця 6.1

Тест-кейси web-платформи AutoHub

№	Функція	Дія користувача	Очікуваний результат	Статус
1	Відкриття головної сторінки	Користувач переходить на головну сторінку	Сторінка завантажується, надоступна навігація до каталогу	Успішно
2	Реєстрація	Користувач вводить email, ім'я та пароль	Створюється новий обліковий запис	Успішно
3	Авторизація	Користувач вводить правильний email і пароль	Система повертає JWT-ітокен, користувач входить у систему	Успішно
4	Помилка авторизації	Користувач вводить неправильний пароль	Система відображає повідомлення про помилку входу	Успішно
5	Перегляд каталогу	Користувач відкриває сторінку каталогу	Відображається список автомобілів	Успішно
6	Фільтрація за маркою	Користувач обирає марку автомобіля	У каталозі залишаються авто відповідної марки	Успішно
7	Фільтрація за ціною	Користувач задає діапазон ціни	Відображаються авто в межах заданої ціни	Успішно

Тест-кейси web-платформи AutoHub

№	Функція	Дія користувача	Очікуваний результат	Статус
8	Перегляд деталей авто	Користувач відкриває картку автомобіля	Відображаються фото, характеристики, ціна та джерело	Успішно
9	Додавання до обраного	Авторизований користувач натискає кнопку додавання	Авто додається до списку обраного	Успішно
10	Видалення обраного	Користувач видаляє авто зі списку	Авто зникає зі сторінки обраного	Успішно
11	Порівняння авто	Користувач обирає 2–5 автомобілів	Відображається таблиця порівняння характеристик	Успішно
12	Перегляд аналітики	Користувач відкриває сторінку аналітики	Відображаються статистичні показники ринку	Успішно
13	Deal Score	Користувач відкриває деталі авто	Система відображає оцінку вигідності пропозиції	Успішно
14	Профіль користувача	Авторизований користувач відкриває профіль	Відображаються дані користувача та персональна інформація	Успішно
15	Адмін-панель	Адміністратор відкриває адміністративну панель	Відображається статистика системи та службові дії	Успішно

За результатами функціонального тестування встановлено, що основні сценарії роботи web-платформи виконуються коректно. Користувач може переглядати каталог, використовувати фільтри, відкривати деталі автомобіля,

додавати пропозиції до обраного та порівнювати авто. Авторизація дозволяє розмежувати доступ до персональних і службових функцій [7].

Окремо перевірено роботу функцій, пов'язаних із даними автомобілів. Система коректно обробляє параметри фільтрації, повертає список автомобілів відповідно до заданих умов і не завершує роботу з помилкою у випадку, якщо результатів пошуку немає. У такій ситуації користувачу має відображатися порожній список або повідомлення про відсутність відповідних оголошень.

Також перевірено роботу сторінки деталей автомобіля. Вона повинна відображати основні характеристики авто, фото, ціну, пробіг, рік випуску, регіон, джерело оголошення та додаткові аналітичні дані. Якщо певне поле відсутнє, система не повинна порушувати структуру сторінки, а має або приховати відповідний блок, або відобразити нейтральне значення.

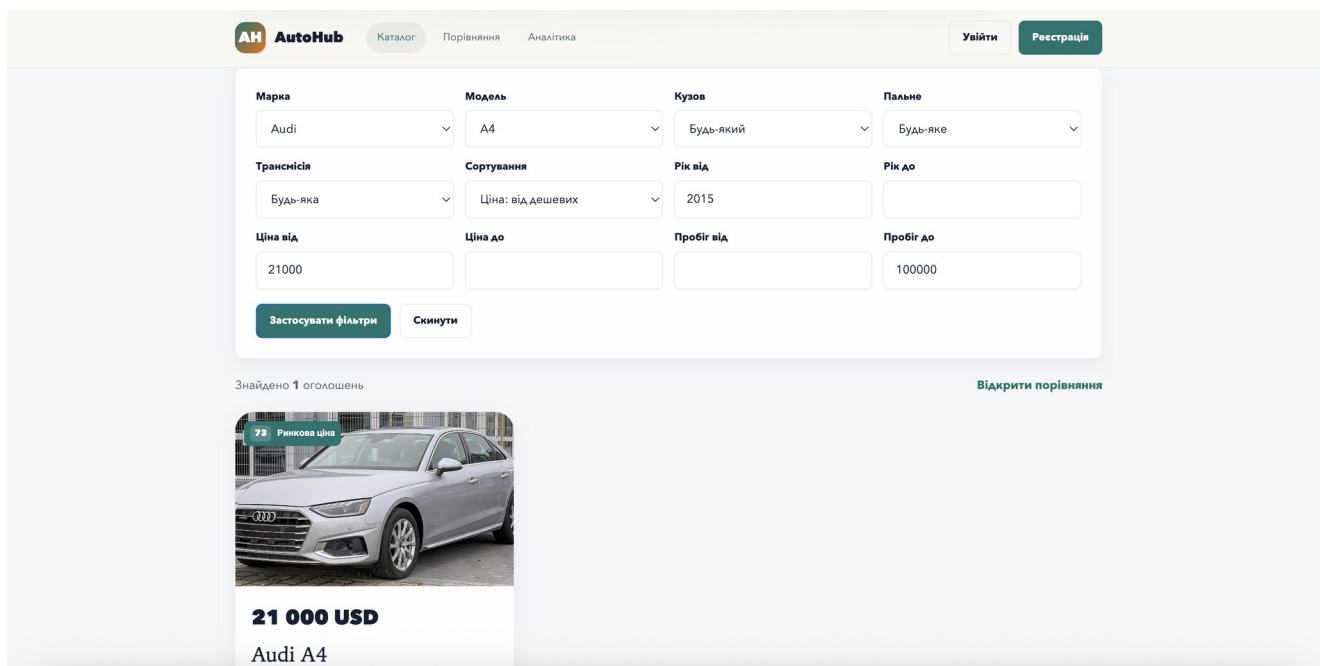


Рис. 6.1 Результат фільтрації автомобілів у каталозі

На рисунку 6.1 показано приклад роботи фільтрації в каталозі автомобілів. Після встановлення параметрів пошуку система повертає лише ті оголошення, які відповідають заданим критеріям. Це дозволяє користувачу швидко звужити список пропозицій і працювати тільки з релевантними результатами.

6.3 Перевірка ролей доступу

Окремим етапом тестування є перевірка ролей доступу. Для web-платформи AutoHub це важливо, оскільки система має публічні, персональні та адміністративні розділи. Якщо доступ не буде обмежений належним чином, звичайний користувач зможе отримати доступ до службових функцій або змінити дані, до яких він не повинен мати доступу.

У системі передбачено три основні сценарії доступу: гість, користувач і адміністратор. Гість має доступ лише до публічних сторінок. Він може переглядати головну сторінку, каталог, сторінку деталей автомобіля та аналітику, якщо вона визначена як публічна. Водночас гість не може додавати авто до обраного, відкривати профіль або адміністративну панель [1 - 6].

Зареєстрований користувач із роллю USER має доступ до персональних функцій. Він може додавати авто до обраного, переглядати збережені автомобілі, порівнювати пропозиції, відкривати профіль і працювати з власними даними. Однак він не має права відкривати адміністративну панель або виконувати службові дії.

Адміністратор із роллю ADMIN має доступ до розширених можливостей. Він може переглядати статистику системи, список користувачів, кількість автомобілів, обраних записів, історію цін, а також запускати оновлення даних. Такі функції повинні бути недоступні для ролі USER.

Перевірка показала, що розмежування доступу відповідає логіці системи. Публічні сторінки доступні всім користувачам, персональні функції потребують авторизації, а адміністративні функції доступні лише користувачу з роллю ADMIN. Такий підхід забезпечує базовий рівень безпеки та дозволяє уникнути несанкціонованого доступу до службових можливостей [7].

На backend-стороні перевірка доступу виконується за допомогою Spring Security та JWT. Коли користувач надсилає запит до захищеного ресурсу, сервер перевіряє токен, визначає користувача та його роль. Якщо роль не відповідає вимогам endpoint, доступ блокується.

На frontend-стороні також реалізується візуальне обмеження доступу. Наприклад, пункт меню адміністративної панелі не повинен відображатися для звичайного користувача. Проте основна перевірка все одно має виконуватися на backend, оскільки frontend-обмеження можна обійти ручним запитом.

6.4 Оцінка роботи механізму Deal Score

Механізм AutoHub Deal Score є однією з ключових особливостей системи, тому його перевірка виконується окремо. Мета тестування полягає у тому, щоб переконатися, що система коректно формує оцінку вигідності автомобільного оголошення та надає користувачу зрозумілий результат [1 - 4].

Під час перевірки враховується, що Deal Score формується на основі кількох факторів: ціни, пробігу, актуальності оголошення, історії зміни ціни та популярності марки або моделі. Кожен фактор робить внесок у загальну оцінку, а підсумковий результат подається у вигляді числового значення та текстової категорії.

Для перевірки механізму було сформовано декілька умовних прикладів автомобілів з різними характеристиками (див. табл. 6.2). Перший автомобіль має ціну нижчу за середню, помірний пробіг і свіже оголошення. Така пропозиція повинна отримати високу оцінку. Другий автомобіль має ринкову ціну та середні характеристики, тому отримує середній бал. Третій автомобіль має завищену ціну або значний пробіг, тому його оцінка повинна бути нижчою.

Таблиця 6.2

Приклад розрахунку AutoHub Deal Score для автомобілів

Автомобіль	Ціна відносно ринку	Пробіг	Актуальність	Історія ціни	Deal Score	Категорія
Toyota Camry 2018	Нижча за середню	Помірний	Свіже оголошення	Є зниження ціни	86	Вигідна пропозиція
Volkswagen Passat 2017	Близька до середньої	Нормальний	Актуальне	Без змін	72	Ринкова ціна
BMW Series 2016	Вища за середню	Високий	Старе оголошення	Немає історії	48	Потребує перевірки
Audi A6 2015	Вища за середню	Дуже високий	Старе оголошення	Без зниження	39	Дорожче ринку

Результати перевірки показують, що Deal Score дозволяє швидко відрізнити потенційно вигідні пропозиції від менш привабливих. Високий бал отримують автомобілі, які мають адекватну або нижчу за ринок ціну, прийнятний пробіг і актуальне оголошення. Нижчий бал отримують авто з високою ціною, великим пробігом або недостатньою кількістю даних.

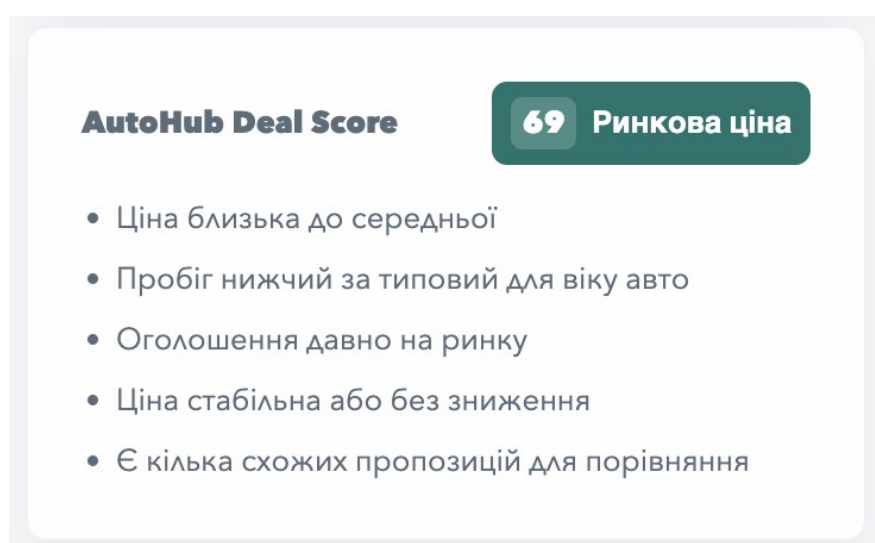


Рис. 6.2 Приклад відображення оцінки Deal Score

На рисунку 6.2 показано приклад відображення оцінки Deal Score на сторінці деталей автомобіля. Користувач бачить не тільки числову оцінку, а й коротке пояснення, чому система віднесла пропозицію до певної категорії. Це робить механізм більш зрозумілим і корисним для практичного використання.

Важливо зазначити, що AutoHub Deal Score не є остаточним рішенням щодо купівлі автомобіля. Він виконує роль аналітичної підказки, яка допомагає користувачу швидше зорієнтуватися у великій кількості оголошень. Остаточне рішення все одно повинно прийматися з урахуванням технічного стану авто, перевірки документів, історії обслуговування та огляду автомобіля.

6.5 Аналіз результатів

Після виконання функціонального тестування, перевірки ролей доступу та оцінки роботи Deal Score можна зробити висновок, що web-платформа AutoHub відповідає основним поставленим вимогам. Система забезпечує централізовану роботу з автомобільними оголошеннями, дозволяє користувачу виконувати пошук, фільтрацію, перегляд деталей, додавання до обраного та порівняння декількох автомобілів [1 - 4].

Аналітичний модуль дозволяє користувачу оцінити загальний стан ринку. Завдяки відображенню середніх цін, кількості оголошень і популярних марок користувач отримує ширший контекст для аналізу. Це особливо корисно у випадках, коли потрібно зрозуміти, чи є конкретна ціна типовою для певної моделі.

Механізм AutoHub Deal Score підвищує практичну цінність системи. Він дозволяє не просто переглядати автомобілі, а отримувати узагальнену оцінку вигідності конкретної пропозиції. Завдяки цьому AutoHub виконує функцію підтримки прийняття рішення, а не лише функцію каталогу.

Перевірка ролей доступу показала, що система коректно розмежовує публічні, персональні та адміністративні функції. Гість має доступ до перегляду основних сторінок, користувач — до персональних можливостей, адміністратор

— до службового керування. Така модель відповідає вимогам безпеки та логіці роботи web-платформи.

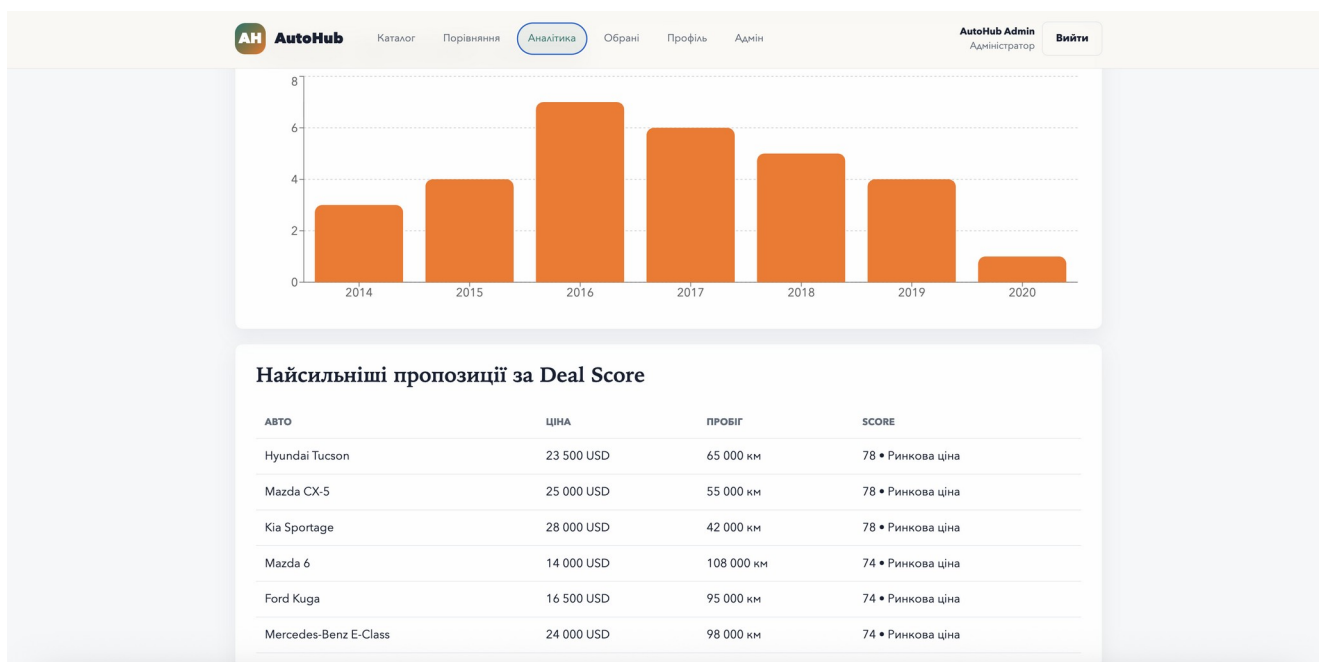


Рис. 6.3 Приклад відображення аналітики ринку

На рисунку 6.3 наведено приклад відображення аналітичної інформації на платформі AutoHub. Аналітичні дані дозволяють користувачу бачити загальні показники ринку, що допомагає краще орієнтуватися у пропозиціях і приймати більш обґрунтовані рішення.

У результаті тестування було підтверджено, що розроблена web-платформа виконує основні функції, визначені в технічних вимогах. Система має зрозумілу структуру, підтримує основні сценарії роботи користувача, забезпечує авторизацію та розмежування доступу, дозволяє працювати з каталогом автомобілів і надає аналітичну підтримку через Deal Score [7].

Реалізоване програмне забезпечення дозволяє спростити процес пошуку, порівняння та оцінювання автомобільних пропозицій, а також створює основу для подальшого розвитку системи, зокрема додавання нових джерел даних, персональних рекомендацій, сповіщень про вигідні пропозиції та розширеної аналітики.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено web-сайт AutoHub для агрегації пропозицій з сайтів продажу автомобілів з пробігом. Розроблена система спрямована на спрощення процесу вибору автомобіля з пробігом і зменшення кількості ручних дій, які користувач зазвичай виконує під час перегляду різних онлайн-платформ.

У першому розділі було проведено аналіз предметної галузі моніторингу ринку вживаних автомобілів. Визначено, що основними проблемами цієї сфери є розміщення оголошень на різних web-платформах, відмінність у структурі подання даних, складність ручного порівняння автомобілів, відсутність єдиного середовища для перегляду пропозицій та недостатня аналітична підтримка користувача.

У другому розділі було здійснено аналіз існуючих програмних рішень для пошуку та моніторингу вживаних автомобілів. Розглянуто сервіси AUTO.RIA, OLX Авто, mobile.de та AutoScout24. У результаті порівняння визначено, що ці платформи мають розвинені засоби пошуку й фільтрації, однак переважно працюють у межах власних баз оголошень.

У третьому розділі було сформульовано постановку задачі розробки та визначено основні вимоги до web-платформи AutoHub. Було визначено об'єкт, предмет і мету роботи, а також сформовано функціональні та нефункціональні вимоги. До основних функцій системи віднесено реєстрацію й авторизацію користувачів, перегляд каталогу автомобілів, пошук і фільтрацію, перегляд детальної картки авто, додавання оголошень до обраного, порівняння автомобілів, перегляд аналітики, роботу з профілем користувача, адміністративне керування та розрахунок показника AutoHub Deal Score. Нефункціональні вимоги охоплюють швидкодію, безпеку, адаптивність інтерфейсу, підтримку одночасної роботи користувачів і збереження значного обсягу оголошень у базі даних.

У четвертому розділі було виконано проектування web-платформи AutoHub. Було розроблено загальну архітектуру системи, описано взаємодію клієнтської та серверної частин, спроектовано структуру бази даних і визначено основні зв'язки між сутностями. Також було спроектовано логіку розрахунку AutoHub Deal Score, яка враховує ціну автомобіля відносно ринку, пробіг, актуальність оголошення, історію зміни ціни та популярність марки або моделі.

У п'ятому розділі було описано програмну реалізацію web-платформи. Серверна частина реалізована з використанням Java, Spring Boot, Spring Data JPA, Spring Security, JWT, PostgreSQL та Liquibase. Клієнтська частина реалізована з використанням React і Vite. Для розгортання та запуску компонентів системи передбачено використання Docker і Docker Compose.

У шостому розділі було проведено тестування web-платформи AutoHub. Перевірено роботу основних користувацьких сценаріїв: реєстрацію, авторизацію, перегляд каталогу, застосування фільтрів, відкриття детальної картки автомобіля, додавання до обраного, порівняння автомобілів, перегляд аналітики, роботу профілю та адміністративної панелі.

Особливістю розробленої web-платформи є механізм AutoHub Deal Score, який дозволяє користувачу швидше оцінити привабливість конкретного автомобільного оголошення. Завдяки цьому AutoHub виконує не лише функцію каталогу, а й функцію підтримки прийняття рішення. Користувач отримує не просто список автомобілів, а додаткову аналітичну підказку щодо того, чи є конкретна пропозиція вигідною, ринковою або такою, що потребує додаткової перевірки.

Отже, мету кваліфікаційної роботи було досягнуто. Розроблена web-платформа AutoHub дозволяє централізувати роботу з оголошеннями про вживані автомобілі, спростити пошук і порівняння пропозицій, надати користувачу аналітичну інформацію та підвищити зручність процесу вибору автомобіля з пробігом.

Робота пройшла апробація. За результатами дослідження опубліковано наступні тези доповіді:

1. Осадчий М.В. Кіберзахист web-сервісу для збору пропозицій з сайтів продажу авто. Матеріали Всеукраїнської науково-практичної конференції «Цифрова трансформація кібербезпеки», 29 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2026. С. 55-56.

2. Осадчий М.В., Шевченко С.В. Веб-сайт для централізованого моніторингу ринку вживаних авто. Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація» 1-3 травня 2026 року. Київ, ДУІКТ. Збірник тез. *(подано до друку)*.

ПЕРЕЛІК ПОСИЛАНЬ

1. Akerlof G. A. The Market for “Lemons”: Quality Uncertainty and the Market Mechanism. *The Quarterly Journal of Economics*. 1970. Vol. 84, no. 3. P. 488–500.
2. Bukvić L., Pavić I., Stamenković M. Price Prediction and Classification of Used-Vehicles Using Supervised Machine Learning. *Sustainability*. 2022. Vol. 14, no. 24. Article 17034. URL: <https://doi.org/10.3390/su142417034>
3. Zhu Y. Prediction of the price of used cars based on machine learning algorithms. *Applied and Computational Engineering*. 2023. Vol. 6, no. 1. P. 785–791. URL: <https://doi.org/10.54254/2755-2721/6/20230917>
4. Nandan M., Sharma P., Singh R. Pre-owned car price prediction by employing machine learning techniques. *Journal of Data Acquisition and Intelligent Computing Systems*. 2023. Vol. 1, no. 1. P. 35–44.
5. Chaffey D., Ellis-Chadwick F. *Digital Marketing*. 8th ed. Harlow: Pearson, 2022. 560 p.
6. Laudon K. C., Traver C. G. *E-commerce 2023–2024: Business, Technology and Society*. 18th ed. Harlow: Pearson, 2023. 912 p.
7. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT): RFC 7519. Internet Engineering Task Force, 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7519>
8. AUTO.RIA: онлайн-майданчик для купівлі та продажу автомобілів. RIA.com. URL: <https://auto.ria.com/> (дата звернення: 26.05.2026).
9. OLX Авто: легкові автомобілі. OLX Україна. URL: <https://www.olx.ua/uk/transport/legkovye-avtomobili/>
10. mobile.de: used cars and new cars marketplace. mobile.de GmbH. URL: <https://www.mobile.de/>
11. AutoScout24: new and used cars marketplace. AutoScout24 GmbH. URL: <https://www.autoscout24.com/>
12. Sommerville I. *Software Engineering*. 10th ed. Boston: Pearson, 2016. 816 p.
13. Pressman R. S., Maxim B. R. *Software Engineering: A Practitioner’s Approach*. 9th ed. New York: McGraw-Hill Education, 2020. 705 p.

14. Keith M., Schincariol M., Nardone M. Pro JPA 2 in Java EE 8: An In-Depth Guide to Java Persistence APIs. 3rd ed. New York: Apress, 2018. 771 p.
15. PostgreSQL Documentation. The PostgreSQL Global Development Group. URL: <https://www.postgresql.org/docs/>
16. Liquibase Documentation. Liquibase Inc. URL: <https://docs.liquibase.com/>
17. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island: Manning Publications, 2019. 520 p.
18. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O'Reilly Media, 2021. 616 p.
19. Walls C. Spring in Action. 6th ed. Shelter Island: Manning Publications, 2022. 520 p.
20. Spring Boot Reference Documentation. VMware, Inc. URL: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>
21. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol: O'Reilly Media, 2020. 310 p.
22. React Documentation. Meta Open Source. URL: <https://react.dev/>
23. Docker Compose Documentation. Docker Inc. URL: <https://docs.docker.com/compose/>
24. Horstmann C. S. Core Java. Volume I: Fundamentals. 12th ed. Boston: Pearson, 2021. 928 p.
25. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley Professional, 2002. 533 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-
КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Розробка web-сайту для агрегації пропозицій з сайтів продажу автомобілів з пробігом

Виконав студент 4 курсу

групи ПД-42

Осадчий Максим Вікторович

Керівник роботи

канд.пед.наук, доцент Шевченко Світлана Миколаївна

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: поліпшення процесу пошуку автомобілів з пробігом за допомогою створення єдиної платформи, яка об'єднує дані з сайтів продажу авто, забезпечуючи можливість фільтрації, порівняння та моніторингу цін.

Об'єкт дослідження: процес пошуку та аналізу пропозицій продажу автомобілів з пробігом в онлайн-середовищі.

Предмет дослідження: розробка web-сайту для агрегації оголошень, фільтрації даних, порівняння характеристик автомобілів з пробігом та представлення аналітики цін.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз літературних джерел і матеріалів за тематикою централізованого моніторингу ринку вживаних автомобілів з пробігом, визначити основні проблеми пошуку, порівняння та оцінювання автомобільних оголошень.
2. Здійснити огляд та аналіз існуючих web-сайтів і сервісів, що використовуються для продажу та моніторингу вживаних автомобілів з пробігом, визначити їх переваги, недоліки та функціональні обмеження.
3. Сформулювати функціональні та нефункціональні вимоги до розробки.
4. Провести огляд засобів для розробки програмного забезпечення веб-застосунку централізованого моніторингу ринку вживаних автомобілів з пробігом.
5. Спроекувати архітектуру web-сайту, структуру бази даних класи та алгоритми обробки.
6. Розробити web-сайт для централізованого збору, пошуку, порівняння та аналітичної обробки даних про вживані автомобілі з пробігом.
7. Провести тестування розробленого web-сайту та оцінити коректність його функціонування і відповідність поставленим вимогам.

3

АНАЛІЗ АНАЛОГІВ

Критерій	AUTO.RIA	AutoTrader	OLX	AutoHub
Пошук вживаних авто з пробігом	Так	Так	Так	Так
Аналітична фільтрація за вигідністю пропозиції	Ні	Ні	Ні	Так
Перевірка історії за VIN кодом	Так	Так	Ні	Так
Порівняння цін	Так	Частково	Ні	Так
Агрегація пропозицій з кількох джерел	Ні	Ні	Ні	Так
Єдина аналітика по різних сайтах	Ні	Ні	Ні	Так

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Збір і відображення оголошень з різних сайтів.
2. Пошук і розширена фільтрація за параметрами авто.
3. Порівняння кількох пропозицій.
4. Відображення історії перевірки та ключових атрибутів.
5. Аналітика цін і виявлення схожих оголошень.
6. Агрегація пропозицій за VIN кодом.
7. Збереження обраних варіантів і сповіщення про нові пропозиції.

Нефункціональні вимоги.

1. Швидкодія пошуку (від 2 до 5 секунд).
2. Масштабованість — можливість підключати додаткові джерела оголошень без надлишкових модифікацій.
3. Хешування паролів для безпеки користувачів (за допомогою bcrypt)
4. Уніфікація даних із різних джерел до єдиної структури.
5. Регулярне (раз на годину) оновлення пропозицій.

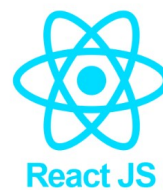
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



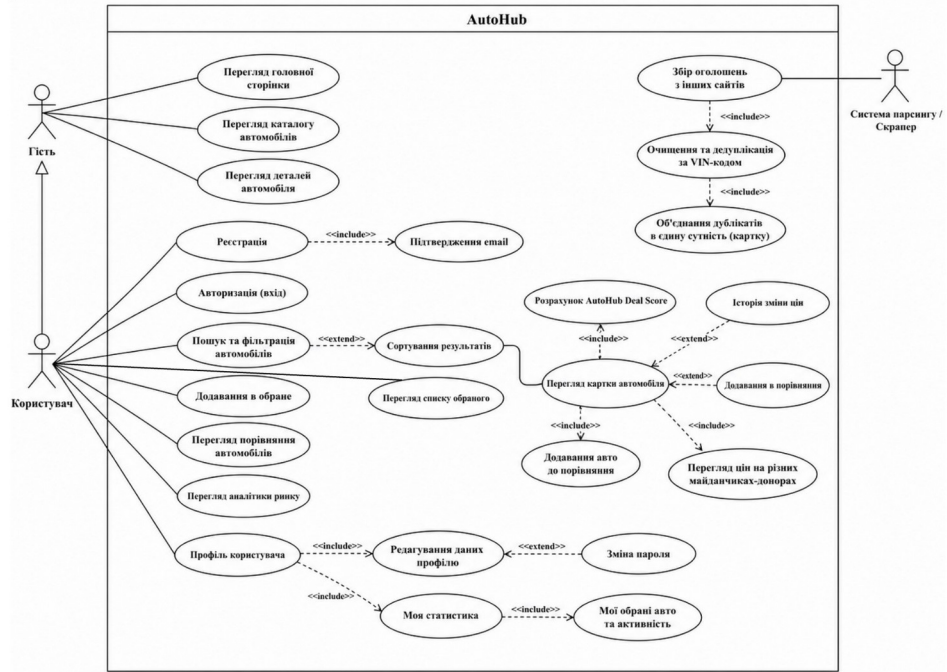
JWT

JSON Web Token



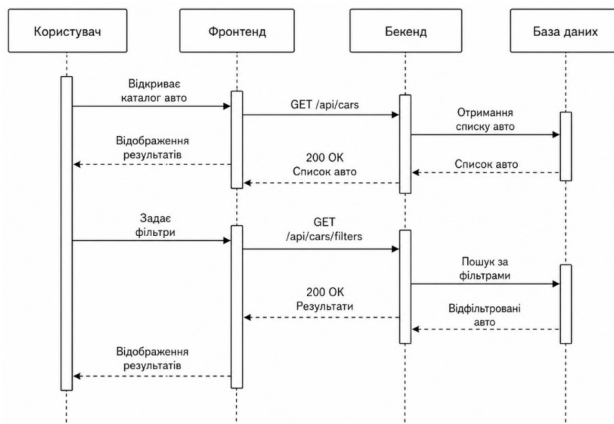
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

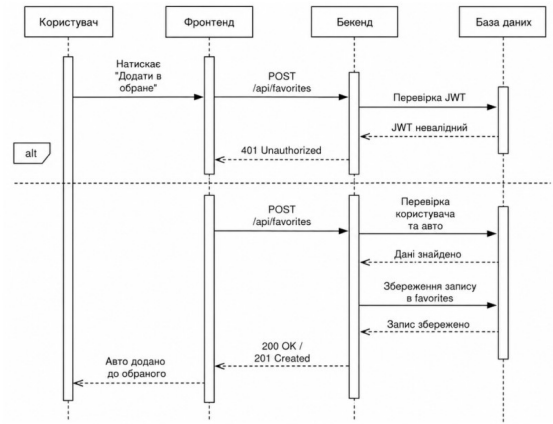


7

ДІАГРАМИ ПОСЛІДОВНОСТІ ОСНОВНИХ ПРЕЦЕДЕНТІВ



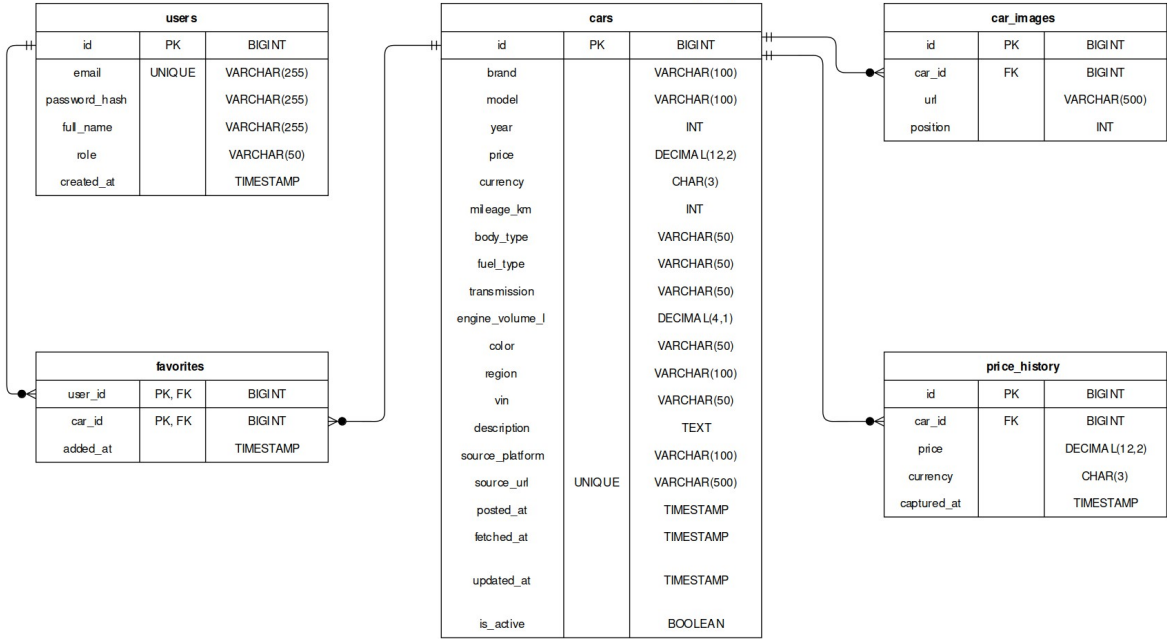
Діаграма послідовності пошуку/фільтрації авто



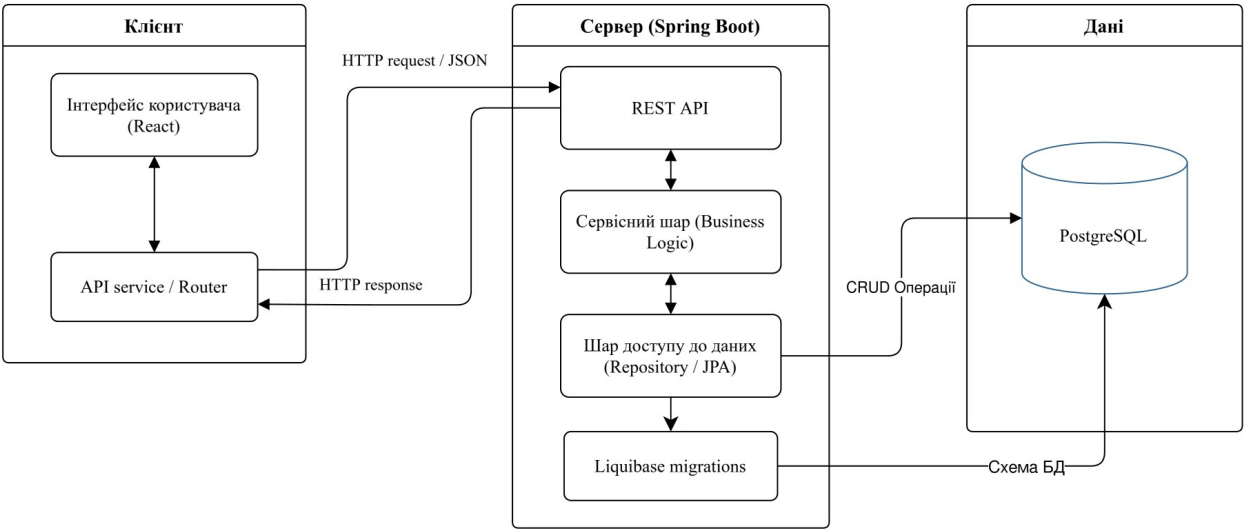
Діаграма послідовності додавання авто до обраного

8

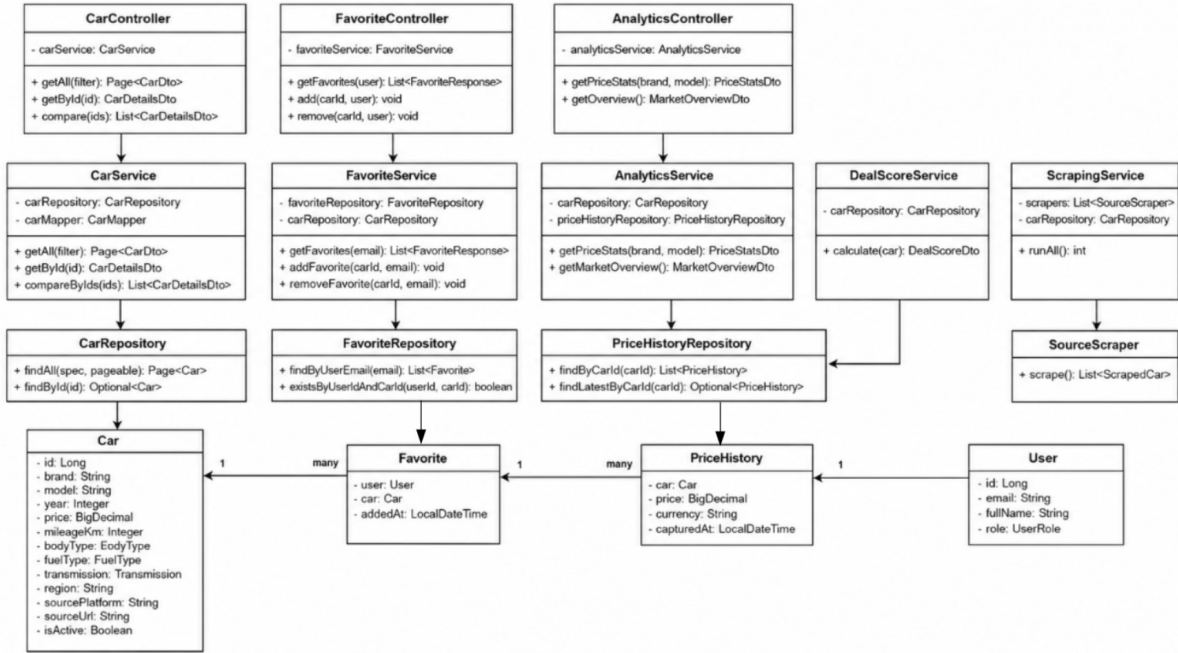
СТРУКТУРА БАЗИ ДАНИХ



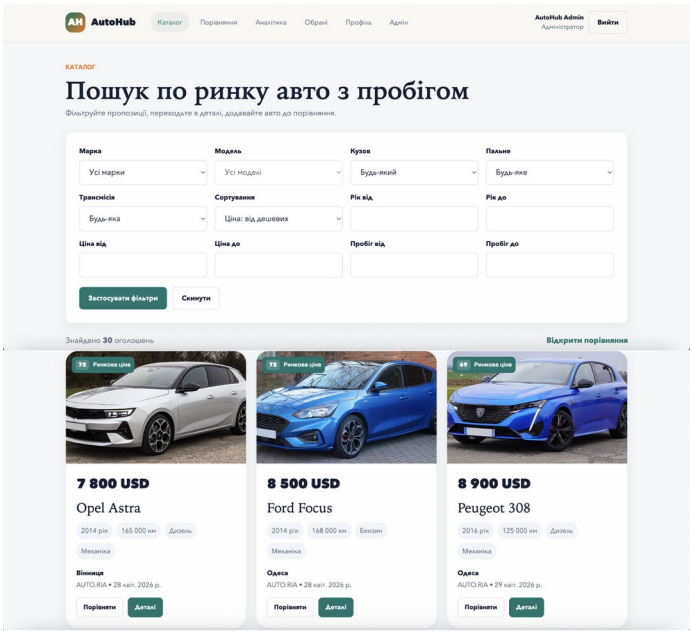
АРХІТЕКТУРА WEB-САЙТУ



ДІАГРАМА КЛАСІВ ОСНОВНОЇ ЛОГІКИ



ЕКРАННІ ФОРМИ



Головна сторінка пошуку автомобілів

ЕКРАННІ ФОРМИ

АН AutoHub

КаталогПорівняння (3)АналітикаОбраніПрофіль

Максим Осадчийuser@autohub.comВийти

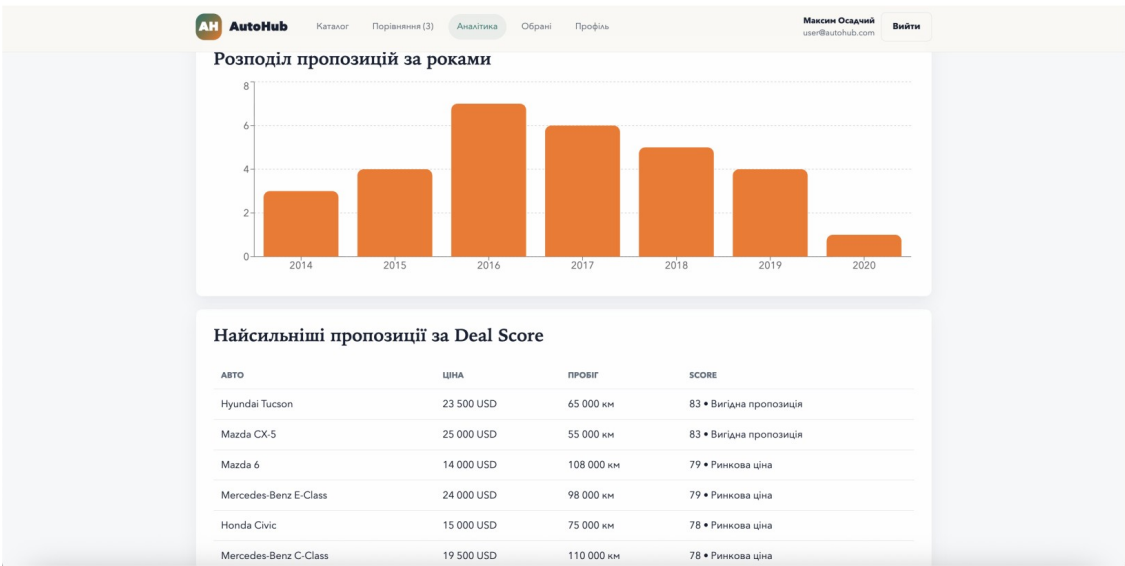
Авто #13 ×Авто #4 ×Авто #15 ×

Параметр	Opel Astra 2014 • 165 000 км Деталі	Ford Focus 2014 • 168 000 км Деталі	Peugeot 308 2016 • 125 000 км Деталі
Ціна	7 800 USD	8 500 USD	8 900 USD
Рік	2014	2014	2016
Пробіг	165 000 км	168 000 км	125 000 км
Кузов	Хетчбек	Хетчбек	Хетчбек
Пальне	Дизель	Бензин	Дизель
Трансмісія	Механіка	Механіка	Механіка
Двигун	1.6 л	1.6 л	1.6 л

Сторінка порівняння авто

13

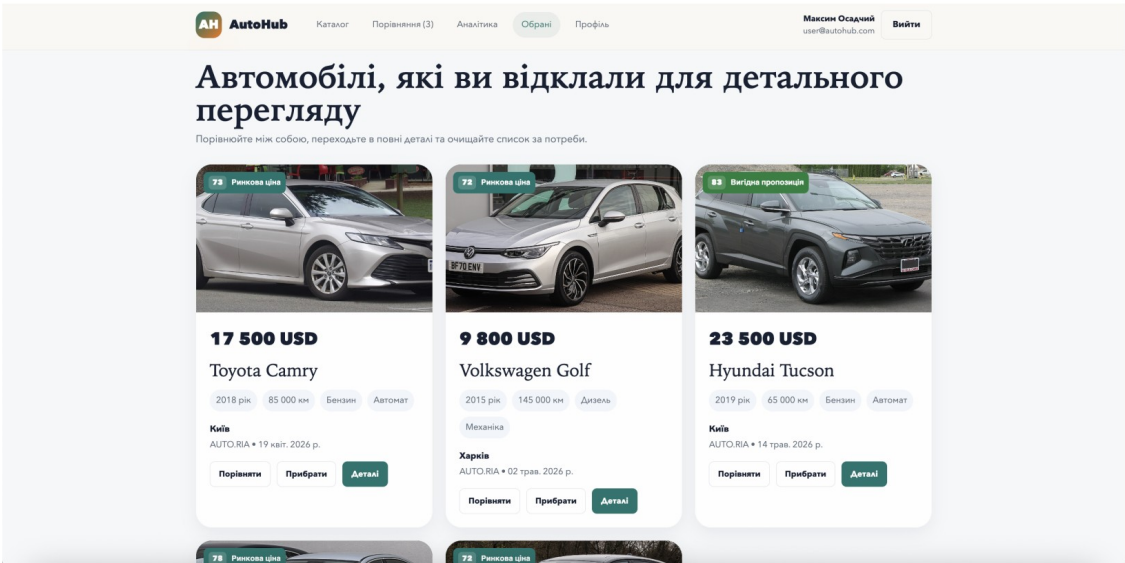
ЕКРАННІ ФОРМИ



Сторінка аналітики оголошень

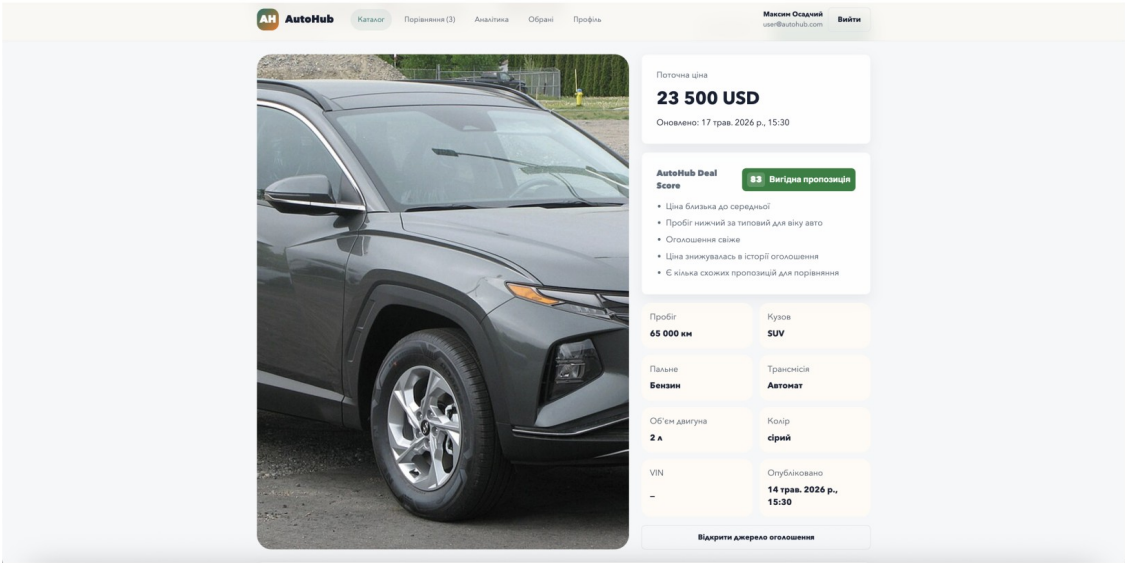
14

ЕКРАННІ ФОРМИ



Сторінка обраних автомобілів

ЕКРАННІ ФОРМИ



Сторінка деталей оголошення

ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ

АН AutoHub Каталог Порівняння Аналітика Обрані Профіль

Максим Осадчий user@autohub.com Вийти

КАТАЛОГ

Пошук по ринку авто з пробігом

Фільтруйте пропозиції, переходьте в деталі, додавайте авто до порівняння.

Марка	Модель	Кузов	Пальне
✓ Усі марки - Audi - BMW - Ford - Honda - Hyundai - Kia - Mazda - Mercedes-Benz - Nissan - Opel - Peugeot - Renault - Skoda	Усі моделі	Будь-який	Будь-яке
Сортування	Рік від	Рік до	
Ціна: від дешевих			
Ціна до	Пробіг від	Пробіг до	

Відкрити порівняння

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

- Осадчий М.В. Кіберзахист web-сервісу для збору пропозицій з сайтів продажу авто. Матеріали Всеукраїнської науково-практичної конференції «Цифрова трансформація кібербезпеки», 29 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2026. С. 55-56.
- Осадчий М.В., Шевченко С.В. Веб-сайт для централізованого моніторингу ринку вживаних авто. Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація» 1-3 травня 2026 року. Київ, ДУІКТ. Збірник тез. (подано до друку).

ВИСНОВКИ

1. Проведено аналіз джерел щодо централізованого моніторингу ринку вживаних автомобілів з пробігом, визначено основні проблеми пошуку, порівняння та оцінювання оголошень. У результаті основними проблемами є розділення інформації між різними джерелами та використання різних форм її представлення.
2. Проаналізовано існуючі веб-сервіси для продажу та моніторингу авто, зокрема AUTO.RIA, AutoTrader та OLX, визначено їх переваги, недоліки та функціональні обмеження, серед яких відсутність функцій агрегації пропозицій з різних джерел та централізованої аналітики.
3. Сформовано функціональні й нефункціональні вимоги, зокрема збір і відображення оголошень з різних сайтів, можливість підключати додаткові джерела оголошень без надлишкових модифікацій та уніфікація даних із різних джерел до єдиної структури.
4. Обґрунтовано вибір засобів розробки web-сайту AutoHub: Java та Spring Boot для серверної частини, PostgreSQL та Liquibase для бази даних, React для браузерної частини, JWT для автентифікації користувачів, та Docker для контейнеризації.
5. Спроектовано архітектуру web-сайту, структуру бази даних та основні діаграми взаємодії компонентів системи.
6. Розроблено web-сайт AutoHub для централізованого збору, пошуку, порівняння й аналітичної обробки даних про вживані автомобілі з функцією оцінювання вигідності пропозиції.
7. Проведено функціональне та нефункціональне тестування системи, яке показало, що застосунок відповідає зазначеним вимогам.

19

ДЯКУЮ ЗА УВАГУ!

20

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Listing 1. AuthController Class

```
package com.example.autohub.controller;

import com.example.autohub.dto.request.LoginRequest;
import com.example.autohub.dto.request.RegisterRequest;
import com.example.autohub.dto.response.AuthResponse;
import com.example.autohub.service.AuthService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
@Tag(name = "Аутентифікація")
public class AuthController {

    private final AuthService authService;

    @PostMapping("/register")
    @ResponseStatus(HttpStatus.CREATED)
    @Operation(summary = "Реєстрація нового користувача")
    public AuthResponse register(@Valid @RequestBody RegisterRequest request) {
        return authService.register(request);
    }

    @PostMapping("/login")
    @Operation(summary = "Вхід у систему")
    public AuthResponse login(@Valid @RequestBody LoginRequest request) {
        return authService.login(request);
    }
}
```

Listing 2. AuthService Class

```
package com.example.autohub.service;

import com.example.autohub.dto.request.LoginRequest;
import com.example.autohub.dto.request.RegisterRequest;
import com.example.autohub.dto.response.AuthResponse;
import com.example.autohub.entity.User;
import com.example.autohub.exception.ConflictException;
import com.example.autohub.repository.UserRepository;
import com.example.autohub.security.JwtTokenProvider;
import lombok.RequiredArgsConstructor;
```

```
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@RequiredArgsConstructor
public class AuthService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtTokenProvider tokenProvider;
    private final AuthenticationManager authenticationManager;

    @Transactional
    public AuthResponse register(RegisterRequest request) {
        if (userRepository.existsByEmail(request.getEmail())) {
            throw new ConflictException("Email вже використовується: " + request.getEmail());
        }

        User user = User.builder()
            .email(request.getEmail())
            .passwordHash(passwordEncoder.encode(request.getPassword()))
            .fullName(request.getFullName())
            .build();

        userRepository.save(user);

        String token = tokenProvider.generateToken(user.getEmail());
        return new AuthResponse(token, user.getEmail(), user.getFullName(), user.getRole().name());
    }

    public AuthResponse login(LoginRequest request) {
        authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(request.getEmail(), request.getPassword())
        );

        User user = userRepository.findByEmail(request.getEmail()).orElseThrow();
    }
}
```

```

        String token =
tokenProvider.generateToken(user.getEmail());
        return new AuthResponse(token, user.getEmail(),
user.getFullName(), user.getRole().name());
    }
}

```

Listing 3. UserController Class

```

package com.example.autohub.controller;

import
com.example.autohub.dto.request.ChangePasswordRequ
est;
import
com.example.autohub.dto.request.UpdateProfileRequest;
import
com.example.autohub.dto.response.ProfileStatsDto;
import
com.example.autohub.dto.response.UserResponse;
import com.example.autohub.service.UserService;
import io.swagger.v3.oas.annotations.Operation;
import
io.swagger.v3.oas.annotations.security.SecurityRequirem
ent;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import
org.springframework.security.core.annotation.Authentica
tionPrincipal;
import
org.springframework.security.core.userdetails.UserDetail
s;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.PatchMapping
;
import
org.springframework.web.bind.annotation.RequestMappi
ng;
import
org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.RestController
;

```

```

@RestController
@RequestMapping("/api/users")
@RequiredArgsConstructor
@Tag(name = "Користувачі")
public class UserController {

    private final UserService userService;

    @GetMapping("/me")
    @Operation(summary = "Профіль поточного
користувача", security = @SecurityRequirement(name =
"bearerAuth"))
    public UserResponse
getMe(@AuthenticationPrincipal UserDetails
userDetails) {

```

```

        return
userService.getMe(userDetails.getUsername());
    }

    @PatchMapping("/me")
    @Operation(summary = "Оновити профіль
поточного користувача", security =
@SecurityRequirement(name = "bearerAuth"))
    public UserResponse
updateMe(@AuthenticationPrincipal UserDetails
userDetails,

        @Valid @RequestBody
UpdateProfileRequest request) {
        return
userService.updateProfile(userDetails.getUsername(),
request);
    }

    @PatchMapping("/me/password")
    @Operation(summary = "Змінити пароль поточного
користувача", security = @SecurityRequirement(name =
"bearerAuth"))
    public void
changePassword(@AuthenticationPrincipal UserDetails
userDetails,

        @Valid @RequestBody
ChangePasswordRequest request) {

        userService.changePassword(userDetails.getUsername(),
request);
    }

    @GetMapping("/me/stats")
    @Operation(summary = "Персональна статистика
поточного користувача", security =
@SecurityRequirement(name = "bearerAuth"))
    public ProfileStatsDto
getMyStats(@AuthenticationPrincipal UserDetails
userDetails) {
        return
userService.getProfileStats(userDetails.getUsername());
    }
}

```

Listing 4. UserService Class

```

package com.example.autohub.service;

import
com.example.autohub.dto.request.ChangePasswordRequ
est;
import
com.example.autohub.dto.request.UpdateProfileRequest;
import
com.example.autohub.dto.response.FavoriteResponse;
import
com.example.autohub.dto.response.ProfileStatsDto;
import
com.example.autohub.dto.response.UserResponse;
import com.example.autohub.entity.Favorite;
import com.example.autohub.entity.User;
import
com.example.autohub.exception.ResourceNotFoundExce
ption;

```

```

import com.example.autohub.mapper.CarMapper;
import
com.example.autohub.repository.FavoriteRepository;
import com.example.autohub.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import
org.springframework.security.authentication.BadCredentialsException;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Comparator;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;
    private final FavoriteRepository favoriteRepository;
    private final PasswordEncoder passwordEncoder;
    private final CarMapper carMapper;

    @Transactional(readOnly = true)
    public UserResponse getMe(String email) {
        User user = getUser(email);
        return toResponse(user);
    }

    @Transactional
    public UserResponse updateProfile(String email,
UpdateProfileRequest request) {
        User user = getUser(email);
        user.setFullName(request.getFullName().trim());
        return toResponse(user);
    }

    @Transactional
    public void changePassword(String email,
ChangePasswordRequest request) {
        User user = getUser(email);
        if (!
passwordEncoder.matches(request.getCurrentPassword()
, user.getPasswordHash())) {
            throw new BadCredentialsException("Невірний
поточний пароль");
        }

        user.setPasswordHash(passwordEncoder.encode(request.
getNewPassword()));
    }

    @Transactional(readOnly = true)
    public ProfileStatsDto getProfileStats(String email) {

```

```

        User user = getUser(email);
        List<Favorite> favorites =
favoriteRepository.findById(user.getId());

        BigDecimal averagePrice = favorites.stream()
            .map(Favorite::getCar)
            .filter(car -> car.getPrice() != null)
            .map(car -> car.getPrice())
            .reduce(BigDecimal.ZERO, BigDecimal::add);

        long pricedCount = favorites.stream().filter(fav ->
fav.getCar().getPrice() != null).count();
        BigDecimal normalizedAverage = pricedCount ==
0
            ? BigDecimal.ZERO
            :
averagePrice.divide(BigDecimal.valueOf(pricedCount),
0, RoundingMode.HALF_UP);

        String topBrand = favorites.stream()
            .collect(Collectors.groupingBy(fav ->
fav.getCar().getBrand(), Collectors.counting()))
            .entrySet()
            .stream()
            .max(Map.Entry.comparingByValue())
            .map(Map.Entry::getKey)
            .orElse("Немає даних");

        List<FavoriteResponse> recentFavorites =
favorites.stream()
            .sorted(Comparator.comparing(Favorite::getAd
dedAt,
Comparator.nullsLast(Comparator.reverseOrder()))
            .limit(5)
            .map(fav -> new
FavoriteResponse(carMapper.toDto(fav.getCar()),
fav.getAddedAt()))
            .toList();

        return new ProfileStatsDto(favorites.size(),
normalizedAverage, topBrand, recentFavorites);
    }

    private UserResponse toResponse(User user) {
        return new UserResponse(user.getId(),
user.getEmail(), user.getFullName(),
user.getRole().name(), user.getCreatedAt());
    }

    private User getUser(String email) {
        return userRepository.findByEmail(email)
            .orElseThrow(() -> new
ResourceNotFoundException("Користувача не
знайдено"));
    }
}

```

Listing 5. CarController Class

```

package com.example.autohub.controller;

import
com.example.autohub.dto.request.CarFilterRequest;

```

```

import
com.example.autohub.dto.response.CarDetailsDto;
import com.example.autohub.dto.response.CarDto;
import com.example.autohub.service.CarService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/cars")
@RequiredArgsConstructor
@Tag(name = "Каталог автомобілів")
public class CarController {

    private final CarService carService;

    @GetMapping
    @Operation(summary = "Список авто з фільтрами та пагінацією")
    public Page<CarDto> getAll(CarFilterRequest filter) {
        return carService.getAll(filter);
    }

    @GetMapping("/{id}")
    @Operation(summary = "Деталі автомобіля")
    public CarDetailsDto getById(@PathVariable Long id) {
        return carService.getById(id);
    }

    @GetMapping("/compare")
    @Operation(summary = "Порівняння кількох автомобілів")
    public List<CarDetailsDto> compare(@RequestParam List<Long> ids) {
        return carService.compareByIds(ids);
    }

    @GetMapping("/brands")
    @Operation(summary = "Список марок для фільтру")
    public List<String> getBrands() {
        return carService.getBrands();
    }

    @GetMapping("/models")
    @Operation(summary = "Список моделей для заданої марки")
    public List<String> getModels(@RequestParam String brand) {
        return carService.getModelsByBrand(brand);
    }
}

```

Listing 6. CarService Class

```

package com.example.autohub.service;

import
com.example.autohub.dto.request.CarFilterRequest;

```

```

import
com.example.autohub.dto.response.CarDetailsDto;
import com.example.autohub.dto.response.CarDto;
import com.example.autohub.entity.Car;
import com.example.autohub.entity.User;
import
com.example.autohub.exception.ResourceNotFoundException;
import com.example.autohub.mapper.CarMapper;
import com.example.autohub.repository.CarRepository;
import
com.example.autohub.repository.FavoriteRepository;
import com.example.autohub.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import
org.springframework.data.jpa.domain.Specification;
import
org.springframework.security.core.Authentication;
import
org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;
import java.util.Set;

@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class CarService {

    private final CarRepository carRepository;
    private final FavoriteRepository favoriteRepository;
    private final UserRepository userRepository;
    private final CarMapper carMapper;

    public Page<CarDto> getAll(CarFilterRequest filter) {
        Specification<Car> spec =
        CarSpecification.build(filter);
        Pageable pageable = buildPageable(filter);
        return carRepository.findAll(spec,
        pageable).map(carMapper::toDto);
    }

    public CarDetailsDto getById(Long id) {
        Car car = carRepository.findById(id)
        .orElseThrow(() -> new
        ResourceNotFoundException("Автомобіль не
        знайдено: " + id));
        boolean isFavorite = checkFavorite(id);
        return carMapper.toDetailsDto(car, isFavorite);
    }

    public List<CarDetailsDto>
    compareByIds(List<Long> ids) {

```

```

        if (ids == null || ids.size() < 2 || ids.size() > 5) {
            throw new IllegalArgumentException("Вкажіть
від 2 до 5 id для порівняння");
        }
        Set<Long> favoriteIds =
getCurrentUserFavoriteIds();
        return carRepository.findAllById(ids).stream()
            .map(car -> carMapper.toDetailsDto(car,
favoriteIds.contains(car.getId())))
            .toList();
    }

    public List<String> getBrands() {
        return carRepository.findAllActiveBrands();
    }

    public List<String> getModelsByBrand(String brand)
    {
        return carRepository.findModelsByBrand(brand);
    }

    private Pageable buildPageable(CarFilterRequest f) {
        Sort.Direction direction =
"DESC".equalsIgnoreCase(f.getDirection())
            ? Sort.Direction.DESC : Sort.Direction.ASC;
        String sortField = switch (f.getSort()) {
            case "year" -> "year";
            case "mileage" -> "mileageKm";
            case "posted" -> "postedAt";
            case "fetched" -> "fetchedAt";
            default -> "price";
        };
        int size = Math.min(Math.max(f.getSize(), 1), 100);
        return PageRequest.of(f.getPage(), size,
Sort.by(direction, sortField));
    }

    private boolean checkFavorite(Long carId) {
        return resolveCurrentUserId()
            .map(uid ->
favoriteRepository.existsByUserIdAndCarId(uid, carId))
            .orElse(false);
    }

    private Set<Long> getCurrentUserFavoriteIds() {
        return resolveCurrentUserId()
            .map(uid ->
Set.copyOfOf(favoriteRepository.findCarIdsByUserId(uid)
))
            .orElse(Set.of());
    }

    private Optional<Long> resolveCurrentUserId() {
        Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
        if (auth == null || !auth.isAuthenticated() ||
"anonymousUser".equals(auth.getPrincipal())) {
            return Optional.empty();
        }
        return
userRepository.findByEmail(auth.getName()).map(User:
.getId());
    }

```

}

Listing 7. FavoriteService Class

```

package com.example.autohub.service;

import
com.example.autohub.dto.response.FavoriteResponse;
import com.example.autohub.entity.Car;
import com.example.autohub.entity.Favorite;
import com.example.autohub.entity.FavoriteId;
import com.example.autohub.entity.User;
import
com.example.autohub.exception.ResourceNotFoundException;
import com.example.autohub.mapper.CarMapper;
import com.example.autohub.repository.CarRepository;
import
com.example.autohub.repository.FavoriteRepository;
import com.example.autohub.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
@RequiredArgsConstructor
public class FavoriteService {

    private final FavoriteRepository favoriteRepository;
    private final UserRepository userRepository;
    private final CarRepository carRepository;
    private final CarMapper carMapper;

    @Transactional(readOnly = true)
    public List<FavoriteResponse> getFavorites(String
email) {
        User user = getUser(email);
        return
favoriteRepository.findByUserId(user.getId()).stream()
            .map(fav -> new
FavoriteResponse(carMapper.toDto(fav.getCar()),
fav.getAddedAt()))
            .toList();
    }

    @Transactional
    public void addFavorite(Long carId, String email) {
        User user = getUser(email);
        Car car = carRepository.findById(carId)
            .orElseThrow(() -> new
ResourceNotFoundException("Автомобіль не
знайдено: " + carId));

        if (!
favoriteRepository.existsByUserIdAndCarId(user.getId()
, carId)) {
            Favorite fav = Favorite.builder()
                .id(new FavoriteId(user.getId(), carId))
                .user(user)
                .car(car)

```

```

        .build();
        favoriteRepository.save(fav);
    }
}

@Transactional
public void removeFavorite(Long carId, String email)
{
    User user = getUser(email);

    favoriteRepository.deleteByUserIdAndCarId(user.getId()
, carId);
}

public boolean isFavorite(Long carId, String email) {
    User user = getUser(email);
    return
    favoriteRepository.existsByUserIdAndCarId(user.getId()
, carId);
}

private User getUser(String email) {
    return userRepository.findByEmail(email)
        .orElseThrow(() -> new
ResourceNotFoundException("Користувача не
знайдено"));
}
}

```

Listing 8. AnalyticsService Class

```

package com.example.autohub.service;

import
com.example.autohub.dto.response.MarketOverviewDto;
import
com.example.autohub.dto.response.MarketOverviewDto.
CountEntry;
import
com.example.autohub.dto.response.PriceHistoryPointDto
;
import
com.example.autohub.dto.response.PriceStatsDto;
import
com.example.autohub.exception.ResourceNotFoundExce
ption;
import com.example.autohub.repository.CarRepository;
import
com.example.autohub.repository.PriceHistoryRepository
;
import
com.example.autohub.repository.projection.PriceStatsPr
ojection;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactiona
l;

import java.util.List;

@Service
@RequiredArgsConstructor

```

```

@Transactional(readOnly = true)
public class AnalyticsService {

    private final CarRepository carRepository;
    private final PriceHistoryRepository
priceHistoryRepository;

    public PriceStatsDto getPriceStats(String brand, String
model) {
        PriceStatsProjection p =
carRepository.findPriceStats(brand, model)
            .orElseThrow(() -> new
ResourceNotFoundException(
                "Немає даних для " + brand + " " +
model));
        return new PriceStatsDto(brand, model,
p.getAvgPrice(), p.getMinPrice(), p.getMaxPrice(),
p.getCount());
    }

    public MarketOverviewDto getMarketOverview() {
        long total = carRepository.countByIsActiveTrue();

        List<CountEntry> byPlatform =
carRepository.findCountByPlatform().stream()
            .map(p -> new CountEntry(p.getPlatform(),
p.getCount()))
            .toList();

        List<CountEntry> byYear =
carRepository.findCountByYear().stream()
            .map(y -> new
CountEntry(String.valueOf(y.getYear()), y.getCount()))
            .toList();

        List<CountEntry> topBrands =
carRepository.findTopBrands(PageRequest.of(0,
10)).stream()
            .map(b -> new CountEntry(b.getBrand(),
b.getCount()))
            .toList();

        return new MarketOverviewDto(total, byPlatform,
byYear, topBrands);
    }

    public List<PriceHistoryPointDto>
getPriceHistory(Long carId) {
        if (!carRepository.existsById(carId)) {
            throw new
ResourceNotFoundException("Автомобіль не
знайдено: " + carId);
        }
        return
priceHistoryRepository.findByCarIdOrderByCapturedAt
Asc(carId).stream()
            .map(ph -> new
PriceHistoryPointDto(ph.getPrice(), ph.getCurrency(),
ph.getCapturedAt()))
            .toList();
    }
}
}

```

Listing 9. ScrapingService Class

```

package com.example.autohub.scrapers;

import com.example.autohub.entity.Car;
import com.example.autohub.entity.CarImage;
import com.example.autohub.entity.PriceHistory;
import com.example.autohub.repository.CarRepository;
import com.example.autohub.repository.PriceHistoryRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.ArrayList;
import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class ScrapingService {

    private final List<SourceScraper> scrapers;
    private final CarRepository carRepository;
    private final PriceHistoryRepository priceHistoryRepository;

    @Scheduled(cron = "0 0 * * *")
    public void scheduleScrap() {
        runAll();
    }

    @Transactional
    public int runAll() {
        int saved = 0;
        for (SourceScraper scraper : scrapers) {
            log.info("Запуск скрейпера: {}",
scraper.getPlatformName());
            try {
                saved += processBatch(scraper.scrape());
            } catch (Exception e) {
                log.error("Помилка скрейпера {}: {}",
scraper.getPlatformName(), e.getMessage());
            }
        }
        log.info("Скрейпінг завершено.
Збережено/оновлено: {} авто", saved);
        return saved;
    }

    private int processBatch(List<ScrapedCarData> items)
    {
        int count = 0;
        for (ScrapedCarData data : items) {
            try {
                count += upsert(data) ? 1 : 0;
            } catch (Exception e) {

```

```

                log.warn("Пропущено запис {}: {}",
data.getSourceUrl(), e.getMessage());
            }
        }
        return count;
    }

    private boolean upsert(ScrapedCarData data) {
        boolean exists =
carRepository.existsBySourceUrl(data.getSourceUrl());
        if (exists) {
            carRepository.findAll().stream()
                .filter(c ->
data.getSourceUrl().equals(c.getSourceUrl()))
                .findFirst()
                .ifPresent(car -> {
                    if (data.getPrice() != null && !
data.getPrice().equals(car.getPrice())) {

priceHistoryRepository.save(PriceHistory.builder()
                    .car(car).price(data.getPrice()).curr
ency(data.getCurrency())
                    .build());
                    car.setPrice(data.getPrice());
                    carRepository.save(car);
                }
            });
            return false;
        }

        Car car = Car.builder()
            .brand(data.getBrand()).model(data.getModel())
            .year(data.getYear())
            .price(data.getPrice()).currency(data.getCurren
cy())
            .mileageKm(data.getMileageKm()).bodyType(
data.getBodyType())
            .fuelType(data.getFuelType()).transmission(dat
a.getTransmission())
            .engineVolumeL(data.getEngineVolumeL()).c
olor(data.getColor())
            .region(data.getRegion()).vin(data.getVin())
            .description(data.getDescription())
            .sourcePlatform(data.getSourcePlatform()).sou
rceUrl(data.getSourceUrl())
            .postedAt(data.getPostedAt())
            .build();

        if (data.getImageUrls() != null) {
            List<CarImage> images = new ArrayList<>();
            for (int i = 0; i < data.getImageUrls().size(); i++)
            {
                images.add(CarImage.builder().car(car).url(data.getImag
eUrls().get(i)).position(i).build());
            }
            car.setImages(images);
        }

        carRepository.save(car);

        if (data.getPrice() != null) {

```

```

priceHistoryRepository.save(PricingHistory.builder()
    .car(car).price(data.getPrice()).currency(data
    .getCurrency())
    .build());
}
return true;
}
}

```

Listing 10. SecurityConfig Class

```

package com.example.autohub.config;

import
com.example.autohub.security.JwtAuthenticationFilter;
import
com.example.autohub.security.UserDetailsService;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;
import
org.springframework.security.authentication.Authentication
Manager;
import
org.springframework.security.authentication.dao.DaoAut
henticationProvider;
import
org.springframework.security.config.annotation.authenti
cation.configuration.AuthenticationConfiguration;
import
org.springframework.security.config.annotation.method.
configuration.EnableMethodSecurity;
import
org.springframework.security.config.annotation.web.buil
ders.HttpSecurity;
import
org.springframework.security.config.annotation.web.con
figuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.con
figurers.AbstractHttpConfigurer;
import
org.springframework.security.config.http.SessionCreati
onPolicy;
import
org.springframework.security.crypto.bcrypt.BCryptPass
wordEncoder;
import
org.springframework.security.crypto.password.Password
Encoder;
import
org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.Userna
mePasswordAuthenticationFilter;
import
org.springframework.web.cors.CorsConfiguration;
import
org.springframework.web.cors.CorsConfigurationSource
;

```

```

import
org.springframework.web.cors.UrlBasedCorsConfigurati
onSource;

```

```

import java.util.List;

```

```

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
@RequiredArgsConstructor
public class SecurityConfig {

```

```

    private final JwtAuthenticationFilter jwtAuthFilter;
    private final UserDetailsService userDetailsService;

```

```

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity
http) throws Exception {
        http
            .csrf(AbstractHttpConfigurer::disable)
            .cors(cors ->
                cors.configurationSource(corsConfigurationSource()))
            .sessionManagement(s ->
                s.sessionCreationPolicy(SessionCreationPolicy.STATEL
ESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(
                    "/api/auth/**",
                    "/api/cars/**",
                    "/api/cars",
                    "/api/analytics/**",
                    "/api-docs/**",
                    "/swagger-ui/**",
                    "/swagger-ui.html"
                ).permitAll()
                .anyRequest().authenticated()
            )
            .authenticationProvider(authenticationProvider
())
            .addFilterBefore(jwtAuthFilter,
UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

```

```

    @Bean
    public CorsConfigurationSource
corsConfigurationSource() {
        CorsConfiguration config = new
CorsConfiguration();

```

```

        config.setAllowedOrigins(List.of("http://localhost:3000",
"http://localhost:5173"));
        config.setAllowedMethods(List.of("GET", "POST",
"PUT", "DELETE", "OPTIONS"));
        config.setAllowedHeaders(List.of("*"));
        config.setAllowCredentials(true);

```

```

        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", config);
        return source;
    }

```

```
}

@Bean
public DaoAuthenticationProvider
authenticationProvider() {
    DaoAuthenticationProvider provider = new
    DaoAuthenticationProvider(userDetailsService);
    provider.setPasswordEncoder(passwordEncoder());
    return provider;
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

@Bean
public AuthenticationManager
authenticationManager(AuthenticationConfiguration
config) throws Exception {
    return config.getAuthenticationManager();
}
}
```