ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій

(повне найменування факультету)

Кафедра комп'ютерних систем та технологій

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «Бакалавр»

«Рекомендаційна інтелектуальна система прокату автомобілів»

(тема кваліфікаційної роботи українською мовою)

«Intelligent car rental recommendation system»

(тема кваліфікаційної роботи англійською мовою)

Виконав: студент денної форми навчання
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)

Освітня програма «Комп'ютерні науки»
(назва освітньої програми)

Люлєв Володимир Володимирович
(прізвище, ім'я, по-батькові)

Керівник к.т.н., доцент Єпік М.О.
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент к.т.н., доцент Камєнєва А.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
_____
№ ___ від ____.____. 20___ р.

Завідувач кафедри
_____ Юрій ГУНЧЕНКО
(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № _____
протокол № __від ____.____.20___ р.

Оцінка_____/___ ___/_____
(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК
_____ Микола МАЛАКСІАНО
(підпис) (ім'я, прізвище)

Одеса 2025

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці рекомендаційної інтелектуальної системи для підбору автомобілів у сфері прокату з використанням алгоритмів машинного навчання.

Мета проєкту – створення ефективного та зручного інструменту, який дозволяє користувачам швидко знайти оптимальний автомобіль для оренди відповідно до їхніх уподобань, історії прокату та інших релевантних параметрів.

У ході роботи проведено аналіз предметної області та існуючих рішень у сфері автопрокату, визначено ключові вимоги до системи та її функціональні можливості. Реалізовано алгоритми машинного навчання для формування персоналізованих рекомендацій на основі поведінки користувача та характеристик транспортних засобів. Спроєктовано архітектуру веб-застосунку, базу даних для зберігання інформації про автомобілі та клієнтів, а також зручний інтерфейс користувача.

Описано вибір технологій для розробки, інтеграцію модулів системи та підходи до навчання моделі. Проведено тестування працездатності системи, зокрема її точність у формуванні релевантних рекомендацій та зручність для користувача.

ABSTRACT

The bachelor's thesis is devoted to the development of an intelligent recommendation system for car rental selection using machine learning algorithms.

The project's goal is to create an effective and convenient tool that allows users to quickly find the best car for rent according to their preferences, rental history, and other relevant parameters.

In the process, we analyzed the subject area and existing solutions in the car rental industry, identified key requirements for the system and its functionality. Machine learning algorithms have been implemented to generate personalized recommendations based on user behavior and vehicle characteristics. The web-application architecture, a database structure for storing information about cars and clients, and a user-friendly interface have been designed.

The choice of development technologies, system module integration, and model training approaches are described. The system's performance was tested, including its accuracy in generating relevant recommendations and user-friendliness.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

UI – інтерфейс користувача (User Interface).

UX – досвід користування (User Experience).

SPA – односторінковий веб-застосунок (Single Page Application).

SQL – структурована мова запитів (Structured Query Language).

BLS – один із провідних українських сервісів прокату автомобілів.

CRM – система управління взаємовідносинами з клієнтами (Customer Relationship Management).

AI – штучний інтелект (Artificial Intelligence).

GPT – генеративно-попередньо навчена трансформер-модель (Generative Pre-trained Transformer).

API – інтерфейс прикладного програмування (Application Programming Interface).

JSON – формат обміну даними, оснований на JavaScript (JavaScript Object Notation).

IDE – інтегроване середовище розробки (Integrated Development Environment).

SSR – серверний рендеринг (Server-Side Rendering).

TS / TypeScript – мова програмування, яка розширює JavaScript системою типів.

Jest – фреймворк для тестування JavaScript/TypeScript застосунків.

Unit-тестування – модульне тестування окремих частин системи.

Functional-тестування – перевірка відповідності роботи компонентів функціональним вимогам.

Supabase – платформа з відкритим кодом для створення бекенду з використанням PostgreSQL.

PostgreSQL – об’єктно-реляційна система керування базами даних.

Node.js – середовище виконання JavaScript на сервері.

React – бібліотека JavaScript для побудови інтерфейсів користувача.

Tailwind CSS – утилітарний CSS-фреймворк для швидкої розробки інтерфейсів.

Framer Motion – бібліотека для анімацій у React-застосунках.

Redux Toolkit – інструментарій для керування станом у React-застосунках.

Frontend – клієнтська частина веб-застосунку.

Backend – серверна частина веб-застосунку.

MySQL – система керування базами даних (змінно до PostgreSQL).

Python – інтерпретована об’єктно-орієнтована мова програмування.

ШІ – штучний інтелект

AI – Artificial Intelligence

ВСТУП

Актуальність теми Сфера прокату автомобілів активно розвивається в умовах зростаючої мобільності населення та популярності оренди як альтернативи володінню транспортом. Водночас зростає потреба у впровадженні сучасних цифрових рішень, які покращують якість обслуговування клієнтів, автоматизують процеси підбору авто та підвищують ефективність управління автопарком. Інтелектуальні системи рекомендацій на основі машинного навчання відкривають нові можливості для персоналізації сервісів, зниження витрат і збільшення клієнтської лояльності.

Мета і завдання проєкту Метою даного проєкту є розробка інтелектуальної системи рекомендацій для підбору автомобілів у прокат, яка забезпечує персоналізовані пропозиції на основі вподобань користувачів, історії оренди, характеристик автомобілів та інших параметрів.

Основними завданнями проєкту є:

1. Провести аналіз предметної області та визначити вимоги до веб-застосунку.
2. Вибрати та адаптувати алгоритми машинного навчання для реалізації персоналізованої рекомендації автомобілів на основі уподобань користувача та історії взаємодій.
3. Спроєктувати структуру та інтерфейс веб-застосунку з урахуванням принципів зручності використання та досвіду користувача.
4. Реалізувати веб-застосунок з використанням сучасних технологій веб-розробки, забезпечивши його швидкодію, масштабованість та безпеку.
5. Інтегрувати базу даних автомобілів та користувачів, з можливістю динамічного оновлення інформації.
6. Протестувати веб-застосунок з використанням різних методів тестування для забезпечення коректної роботи та зручності використання.

7. Розробити інструкцію з використання веб-застосунку для кінцевих користувачів.

Об’єктом дослідження є процес підбору автомобілів для клієнтів у сервісах прокату.

Предметом дослідження є веб-застосунок для рекомендації автомобілів з використанням алгоритмів машинного навчання.

Методи дослідження У процесі розробки веб-застосунку буде використано наступні методи дослідження:

1. Огляд наукових праць, аналітичних статей та інтернет-джерел для вивчення предметної області та існуючих рішень у сфері прокату авто і рекомендаційних систем.
2. Використання моделей машинного навчання, таких як колаборативна та контентна фільтрація, для створення рекомендаційного механізму.
3. Моделювання структури системи з використанням діаграм UML (Use Case, Activity, Class).
4. Розробка інтерфейсу користувача на основі принципів UI/UX дизайну.
5. Імплементація веб-застосунку з використанням JavaScript/TypeScript, React, Node.js, бази даних MySQL.
6. Проведення модульного, інтеграційного, функціонального та юзабіліті тестування.
7. Збір зворотного зв’язку від тестових користувачів з подальшим аналізом для покращення функціональності системи.

Практична цінність проєкту Розроблюваний веб-застосунок дозволяє компаніям з прокату автомобілів покращити взаємодію з клієнтами, оптимізувати процес підбору автомобіля, знизити навантаження на персонал та підвищити точність відповідності між потребами клієнтів і запропонованими авто. Впровадження такої системи сприяє підвищенню якості сервісу, конкурентоспроможності компанії та ефективному використанню автопарку.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ І ОГЛЯД ПІДХОДІВ ДО ВИРІШЕННЯ ЗАВДАННЯ

1.1 Опис предметної області

Предметна область даного проєкту охоплює сферу прокату автомобілів – одну з найдинамічніших галузей транспортного сервісу, яка надає користувачам тимчасовий доступ до автомобілів з метою задоволення їхніх мобільних потреб без необхідності володіння транспортним засобом [1].

Прокат авто є особливо актуальним для туристів, ділових поїздок, короткострокового користування або у випадках, коли власний автомобіль недоступний.

З огляду на зростання попиту на такі послуги, компанії, що займаються автопрокатом, стикаються з необхідністю автоматизації основних бізнес-процесів: управління автопарком, обробки запитів клієнтів, бронювання авто, а також покращення якості обслуговування. Один із ключових викликів – забезпечення швидкого та точного підбору автомобіля, який найкраще відповідає потребам клієнта за такими критеріями, як тип авто, бюджет, тривалість оренди, розмір, трансмісія, паливна ефективність, додаткові функції тощо [2].

Традиційні методи підбору передбачають ручний перегляд доступного транспорту або фільтрацію за основними параметрами, що не завжди дозволяє врахувати повний контекст користувацьких уподобань. Це може призвести до низького рівня задоволеності клієнтів, втрати часу, а також меншої ефективності використання парку авто.

У зв’язку з цим актуальною є побудова рекомендаційної системи, яка на основі аналізу історії користування, вподобань клієнта та характеристик автопарку може автоматично пропонувати найбільш релевантні варіанти. Такі системи широко використовуються в електронній комерції, сервісах

потокового відео та інтернет-магазинах, демонструючи високу ефективність у підвищенні конверсій та задоволеності користувачів [3].

У контексті прокату автомобілів інтелектуальна система має враховувати не лише поточні параметри запиту, а й складні поведінкові шаблони, сезонні зміни, попередні вибори клієнта та статистику користування подібними автомобілями іншими користувачами. У результаті компанія отримує інструмент, що не тільки покращує якість обслуговування, але й підвищує прибутковість за рахунок оптимізації розподілу автопарку та зниження кількості незадіяних машин.

Таким чином, предметна область охоплює як технічні аспекти реалізації інформаційної системи, так і бізнес-компонент: облік даних про автомобілі, поведінковий аналіз клієнтів, автоматизований підбір варіантів, інтеграцію з CRM та забезпечення зручного інтерфейсу для кінцевого користувача.

1.2 Аналіз існуючих рішень

На сучасному ринку існує низка веб-застосунків і мобільних сервісів, що надають послуги оренди автомобілів, а також певні функції пошуку та фільтрації доступного транспорту. Проте більшість із них не використовують повноцінні рекомендаційні системи з елементами штучного інтелекту або машинного навчання, обмежуючись лише базовими інструментами сортування та категоризації.

Одним із найвідоміших міжнародних сервісів є Rentalcars.com – платформа, яка дозволяє користувачам орендувати автомобілі в різних країнах і містах (рис. 1.2.1). Вона пропонує фільтри за типом авто, передбачуваною ціною, класом, умовами прокату тощо. Однак система не аналізує поведінкові шаблони користувачів, не зберігає історію їхніх попередніх оренд і не надає персоналізованих рекомендацій [4].

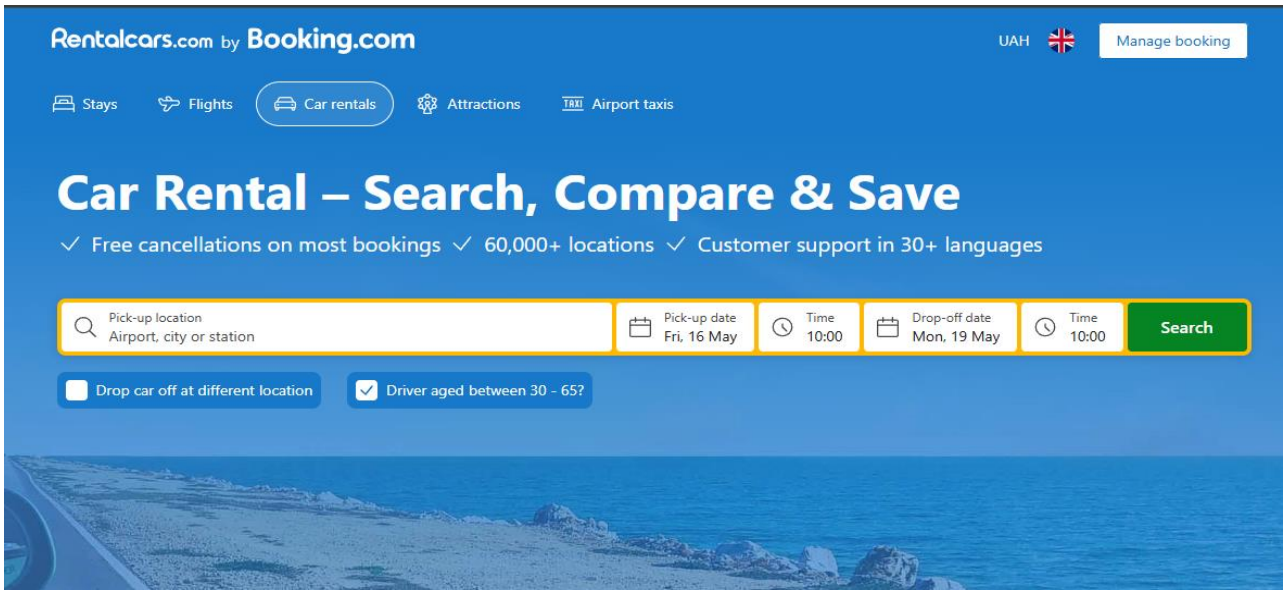


Рисунок 1.2.1 – Веб-сервіс Rentalcars

Getaround – платформа з елементами peer-to-peer прокату, яка надає можливість орендувати авто напряму у власників (рис. 1.2.2). Вона має сучасний інтерфейс та функцію швидкого пошуку, однак також не впроваджує складних рекомендаційних алгоритмів [5].

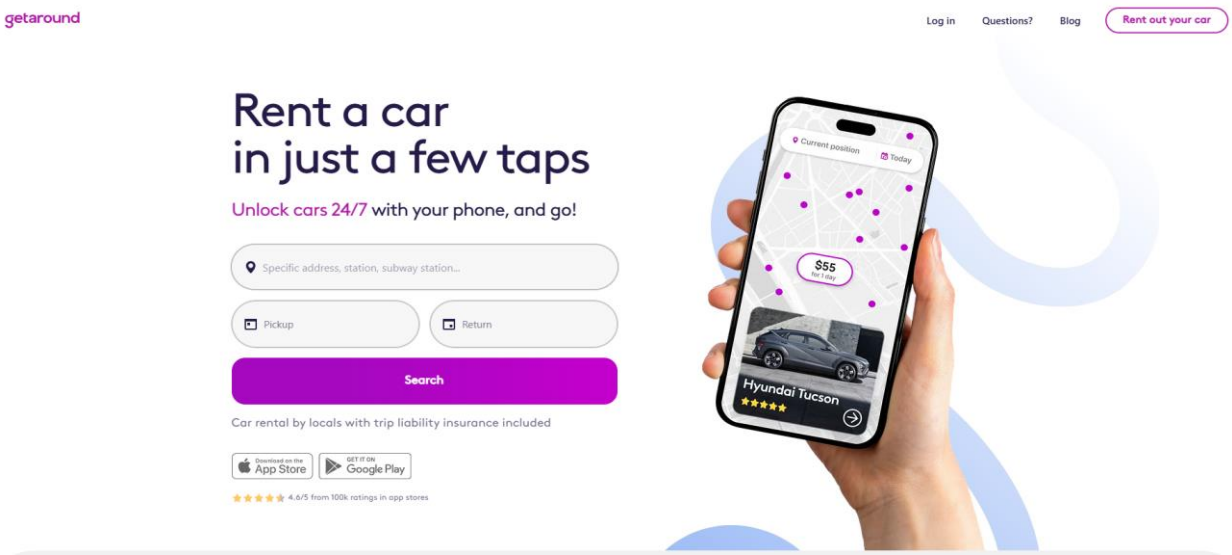


Рисунок 1.2.2 – Веб-сервіс Getaround

На українському ринку популярністю користується сервіс BLS – національний оператор прокату автомобілів, що надає послуги оренди в Києві, Львові, Одесі, Харкові, Дніпрі та інших містах України (рис. 1.2.3). Користувачі можуть обрати авто із запропонованого переліку, застосовуючи фільтри за категоріями (економ, стандарт, преміум), типом кузова, трансмісією, паливом тощо. Однак, подібно до інших сервісів, BLS не враховує індивідуальні вподобання користувача або його історію оренди. Вибір відбувається вручну, без інтелектуальної підтримки [6].

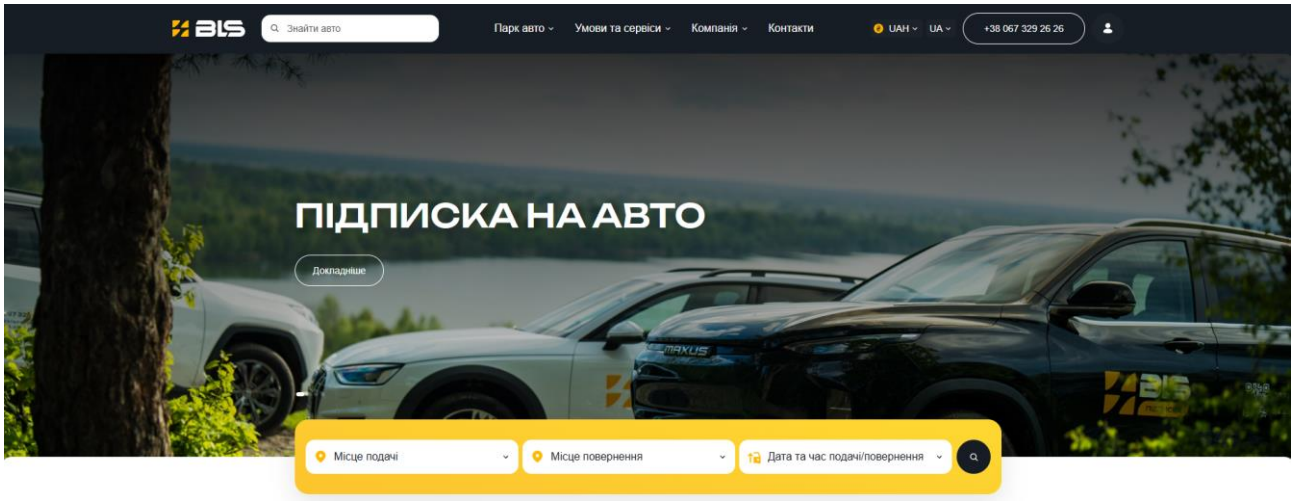


Рисунок 1.2.3 – Веб-сервіс BLS

Загальною рисою всіх зазначених платформ є відсутність адаптивного механізму, який би автоматично аналізував дані користувача та надавав йому персоналізовані рекомендації з урахуванням його минулого досвіду, вподобань, а також реальних показників автопарку (наприклад, рейтинг авто, частота оренди, ефективність, сезонність тощо).

На відміну від існуючих рішень, пропонована система має на меті інтеграцію алгоритмів машинного навчання, які дозволять не лише автоматизувати підбір автомобіля, а й забезпечити більш точний, зручний та індивідуалізований підхід до кожного клієнта. Такий підхід є інноваційним у

сфері автопрокату, оскільки дозволяє підвищити ефективність використання автопарку, зменшити час пошуку авто та покращити якість обслуговування.

1.3 Обґрунтування актуальності розробки

Розробка веб-застосунку для рекомендації автомобілів у сфері прокату з використанням алгоритмів машинного навчання є актуальним і перспективним напрямом, який відповідає сучасним тенденціям у галузі цифрової трансформації сервісів, орієнтованих на клієнта.

Актуальність проєкту зумовлена кількома важливими факторами. По-перше, ринок оренди автомобілів демонструє стале зростання, зокрема в міських центрах, серед туристів, ділових мандрівників і користувачів, які не потребують постійного володіння авто. Попит на мобільність збільшується, як і очікування щодо якості та швидкості обслуговування. Користувачі бажають швидко отримувати персоналізовані пропозиції без необхідності вручну переглядати десятки варіантів.

По-друге, більшість сучасних сервісів прокату авто працюють за схемою класичного фільтрування, надаючи лише поверхневу можливість сортування за базовими параметрами (тип кузова, трансмісія, ціна тощо). При цьому упущено важливий аспект – врахування поведінкової історії користувача, його попередніх уподобань, частоти оренди, вподобаних моделей або умов прокату. Це створює прогалину в індивідуалізації сервісу, яку можливо заповнити саме за рахунок рекомендаційної системи.

По-третє, розвиток алгоритмів машинного навчання відкриває нові можливості в автоматизованій обробці даних клієнтів, сегментації користувачів, передбаченні їхніх запитів і побудові точних рекомендацій. Це дозволяє створити не просто платформу для оренди авто, а інтелектуальну систему, що «розуміє» клієнта, адаптується до його потреб та навчається з часом.

Також варто зазначити, що застосування рекомендаційних систем дозволяє підвищити ефективність використання автопарку. За рахунок точного розподілу попиту, врахування сезонності та популярності моделей, компанії можуть краще планувати закупівлі, зменшити кількість простою транспорту, а також покращити логістику.

З боку кінцевого користувача, запропонований веб-застосунок дозволяє:

1. Зекономити час на пошук і порівняння варіантів.
2. Отримати персоналізовані рекомендації, які відповідають попередньому досвіду та бюджету.
3. Довіритися системі, що автоматично підбере авто відповідно до потреб (наприклад, для поїздки з дітьми, ділової зустрічі, подорожі в гори тощо).
4. Мати доступ до інтуїтивно зрозумілого інтерфейсу з мобільних або десктопних пристроїв.

З точки зору бізнесу, впровадження такої системи:

1. Покращує конверсію (від перегляду до бронювання).
2. Підвищує лояльність клієнтів.
3. Скорочує потребу в консультаціях менеджера.

Забезпечує конкурентну перевагу на ринку прокату.

Крім того, дана система має освітній потенціал, оскільки може стати основою для розробки подібних IT-продуктів у транспортній галузі, інтегруватися в міські транспортні сервіси, використовуватися в навчальному процесі для демонстрації роботи рекомендаційних алгоритмів.

Таким чином, розробка рекомендаційної інтелектуальної системи прокату автомобілів є своєчасною, практично значущою та інноваційною. Вона поєднує в собі технології веб-розробки, машинного навчання, UX-дизайну та системної інтеграції, що дозволяє досягти високого рівня автоматизації та персоналізації у сфері автопрокату.

1.4 Постановка завдання

Метою даного проєкту є розробка інтелектуального веб-застосунку для прокату автомобілів, який дозволить користувачам швидко та ефективно підбирати оптимальний автомобіль для оренди відповідно до їхніх уподобань, попереднього досвіду, бюджету та інших релевантних параметрів. Для досягнення цієї мети необхідно вирішити низку ключових завдань.

Проаналізувати предметну область, вивчити особливості ринку прокату автомобілів, типові вимоги клієнтів та проблеми, які виникають при підборі транспорту. Важливим є дослідження наявних цифрових рішень та підходів до рекомендації авто, зокрема з точки зору використання алгоритмів машинного навчання.

Ключовим завданням є визначення критеріїв, які враховуватимуться під час формування рекомендацій. До таких критеріїв можуть входити:

1. Тип поїздки (короткострокова, тривала, сімейна, ділова, туристична).
2. Клас автомобіля (економ, середній, преміум).
3. Бюджет користувача на оренду.
4. Особисті вподобання: тип трансмісії, тип пального, розмір багажника, наявність кондиціонера, мультимедіа тощо.
5. Історія попередніх оренд користувача.
6. Відгуки та рейтинг автомобіля серед інших користувачів.
7. Доступність автомобіля у потрібний період та місці.

Розробити структуру та логіку рекомендаційного алгоритму, зокрема за допомогою підходів машинного навчання, таких як колаборативна фільтрація, контентна фільтрація або їхня комбінація. Система має вміти враховувати історію дій користувача та оновлювати рекомендації з урахуванням змін у запитах.

Наступним кроком є проєктування архітектури веб-застосунку, що включає визначення його основних компонентів: інтерфейсу користувача,

серверної логіки, рекомендаційного модуля, бази даних і механізмів інтеграції з зовнішніми джерелами (наприклад, для отримання інформації про автопарк).

Розробити базу даних, яка міститиме інформацію про:

1) автомобілі (модель, марка, рік випуску, параметри, рейтинг, фото);
2) користувачів та їх історію оренди;
3) правила прокату, доступність авто у конкретні дати;
4) оцінки, коментарі, результати взаємодії з системою.

Провести реалізацію веб-застосунку з використанням сучасних технологій веб-розробки. Клієнтська частина має забезпечувати швидку і зручну взаємодію, а серверна — ефективну обробку запитів та генерацію рекомендацій. Для цього буде використано технології, як React/TypeScript для Frontend, Node.js/Python для Backend, MySQL або іншу СКБД.

Важливим етапом є тестування системи, зокрема:

1) модульне тестування окремих компонентів;
2) інтеграційне тестування взаємодії між модулями;
3) функціональне тестування логіки рекомендацій;
4) UX-тестування з боку користувача.

Важливим етапом є створення докладної інструкції з використання веб-застосунку для кінцевих користувачів, яка пояснюватиме принцип роботи, навігацію по інтерфейсу та способи отримання рекомендацій.

Успішне виконання поставлених завдань дозволить створити сучасний, інтуїтивно зрозумілий і технологічно просунутий веб-застосунок, який поєднує зручність традиційного онлайн-сервісу з можливостями штучного інтелекту.

Така система підвищить рівень автоматизації в сфері прокату автомобілів, покращить користувацький досвід і сприятиме ефективнішому використанню автопарку.

Висновки до розділу 1

У розділі розглянуто предметну область прокату автомобілів, визначено основні проблеми та потреби користувачів, пов’язані з підбором авто для оренди. Проаналізовано існуючі рішення на ринку цифрових сервісів оренди автомобілів, виявлено їхні переваги та обмеження, зокрема відсутність персоналізованого підходу до вибору авто.

Обґрунтовано актуальність створення веб-застосунку з використанням алгоритмів машинного навчання для автоматизованої рекомендації автомобілів. Такий підхід дозволяє враховувати індивідуальні вподобання клієнтів, їхню історію оренд, бюджет, тип поїздки та інші параметри. Інтелектуальна система рекомендацій забезпечить більш гнучкий, швидкий та ефективний процес підбору авто, що сприятиме підвищенню рівня задоволеності користувачів і оптимізації використання автопарку.

Сформульовано мету кваліфікаційної роботи та визначено основні завдання, які необхідно реалізувати для досягнення цієї мети. Це охоплює аналіз предметної області, проектування структури системи, розробку бази даних, створення рекомендаційного модуля, реалізацію інтерфейсу користувача, тестування та підготовку інструкцій з використання.

Загалом, розробка інтелектуального веб-застосунку для рекомендації автомобілів у сфері прокату є актуальним, інноваційним та практично необхідним проєктом, що поєднує сучасні технології веб-розробки та машинного навчання для покращення клієнтського досвіду в автомобільному сервісі.

2 ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНКУ

2.1 Вимоги до інтелектуальної системи прокату автомобілів

Загальна характеристика системи

Інтелектуальна система прокату автомобілів є веб-орієнтованим програмним забезпечення, що дозволяє користувачам здійснювати пошук, фільтрацію, бронювання транспортних засобів та отримувати персоналізовані рекомендації на основі аналізу вподобань і попередньої поведінки. Система передбачає наявність адміністративного модуля для управління автопарком, аналітики замовлень та підтримки актуальної інформації про транспортні засоби.

Основною особливістю системи є використання елементів штучного інтелекту, зокрема алгоритмів машинного навчання, для формування персоналізованих рекомендацій користувачам, що дозволяє підвищити зручність взаємодії з системою, ефективність використання автопарку та рівень задоволеності клієнтів.

Функціональні вимоги

1. Ефективність системи пошуку автомобілів, що дозволяє користувачам здійснювати пошук транспортних засобів за такими параметрами, як бренд, модель, тип кузова, рік випуску, коробка передач, тип пального, клас (економ, бізнес, преміум) тощо. Крім того, користувач має змогу фільтрувати результати за ціною, рейтингом, рівнем доступності в задані дати, що значно спрощує вибір автомобіля.
2. Наявність модулю персоналізованих рекомендацій, який працює на основі алгоритмів машинного навчання. Система аналізує історію бронювань, вподобання користувача, бюджет та контекстні фактори (місцезнаходження, сезон, тривалість оренди) для формування списку рекомендованих автомобілів. Рекомендації мають адаптуватися до змін у поведінці користувача та бути персоніфікованими.

3. Наявність модулю перевірки доступності автотранспорту у реальному часі, що дозволяє системі оперативно визначати, чи доступний конкретний автомобіль у задані дати. У випадку відсутності вільного транспорту система повинна автоматично виключати відповідні варіанти з можливості бронювання.
4. Реєстрація та автентифікації користувачів, що забезпечує створення облікових записів з підтвердженням електронної пошти, входом через логін/пароль або через сторонні сервіси (OAuth: Google, Facebook тощо). Особистий кабінет користувача має містити інформацію про попередні бронювання, обрані налаштування, збережені уподобання та історію відгуків.
5. Можливість оформлення бронювання, яка реалізується через зручну форму, що дозволяє обрати конкретний автомобіль, задати дати початку та завершення прокату, додати додаткові послуги (наприклад, дитяче крісло, навігатор, додаткове страхування). Після цього система повинна автоматично розрахувати вартість прокату, враховуючи усі додаткові платежі, знижки або акції. Користувачу надсилається підтвердження бронювання на email.
6. Наявність адміністративної панелі управління автопарком, яка доступна лише для уповноважених осіб, дозволяє здійснювати додавання, редагування та видалення записів про автомобілі, оновлювати їхні технічні характеристики, доступність і ціни. Окрім того, адміністратор має змогу переглядати звіти про бронювання, аналізувати динаміку попиту, керувати користувачами та їхніми правами доступу.
7. Система рейтингу та відгуків, за допомогою якої користувачі можуть залишати оцінки (у форматі зірок) та текстові коментарі після завершення оренди. Ці оцінки впливають на рейтинг автомобіля та можуть бути використані в алгоритмах рекомендацій.
8. Інтеграція з платіжними системами для онлайн-оплати послуг через популярні платіжні платформи (наприклад, LiqPay, Stripe, PayPal, Apple

Pay). Система також повинна підтримувати функцію передоплати або бронювання з оплатою при отриманні автомобіля.

Нефункціональні вимоги

1. Забезпечення високої продуктивності системи, зокрема часу відгуку на запити користувачів, що не повинен перевищувати 2 секунд при умовах навантаження до 1000 одночасних сесій. Для цього передбачено використання хешування результатів, оптимізацію запитів до бази даних і розподілену обробку.
2. Архітектура системи з використанням мікросервісного підходу із можливістю гнучкого розгортання у хмарних середовищах, таких як AWS, Google Cloud Platform або Heroku. Це забезпечує безперебійну роботу системи при збільшенні кількості користувачів.
3. Зручний та інтуїтивно зрозумілий інтерфейс користувача, що гарантує комфортну роботу на різних пристроях (смартфони, планшети, десктопи). Система має відповідати сучасним вимогам до UX/UI, включаючи адаптивний дизайн, зручну навігацію та підтримку мультимовності.
4. Належний рівень безпеки для захисту даних користувачів та запобігання несанкціонованому доступу, зокрема шифрування даних користувачів, автентифікація через JWT або OAuth 2.0, захист від поширених атак (XSS, CSRF, SQL-ін'єкції). Регулярне резервне копіювання даних має забезпечити відновлення системи у випадку збоїв.
5. Інтероперабельність та інтеграційна сумісність, що досягається завдяки відкритому RESTful API, який дозволяє підключати сторонні сервіси (мобільні додатки, CRM-системи, аналітичні платформи). Передбачається використання форматів обміну даними JSON та підтримка WebSocket для роботи в режимі реального часу.
6. Гарантована надійність і доступність системи, що передбачає безперебійну роботу веб-застосунку в режимі 24/7 із рівнем сервісного

обслуговування (SLA) не нижче 99,9%. Усі помилки повинні оброблятись з відповідним інформуванням користувача.

7. SEO-оптимізація інтерфейсу та контенту, що містить наявність мета-тегів, мапи сайту, розмітки Open Graph для соціальних мереж, а також оптимізацію HTML-структури для підвищення індексації пошуковими системами.
8. Документування системи на всіх етапах розробки: технічна документація для API, керівництво користувача для клієнтів та адміністраторів, а також інструкції з оновлення та супроводу системи.

Цей розширений перелік функціональних та нефункціональних вимог забезпечить створення ефективного, зручного та масштабованого інтелектуального веб-застосунку з прокату автомобілів, який відповідатиме сучасним технологічним стандартам та очікуванням користувачів.

Ретельно визначені функціональні вимоги дозволяють реалізувати усі ключові можливості системи, починаючи від пошуку автомобіля й завершуючи повним циклом бронювання та інтерактивною взаємодією з користувачем. Наявність інтелектуального модуля рекомендацій дозволяє не лише автоматизувати вибір транспортного засобу, а й зробити його персоналізованим, що значно підвищує рівень задоволеності клієнтів і конкурентоспроможність сервісу.

У свою чергу, нефункціональні вимоги гарантують надійність, доступність та продуктивність системи за умов реального навантаження. Забезпечення безпеки, підтримка адаптивного дизайну, кросбраузерна сумісність і можливість інтеграції з іншими сервісами створюють фундамент для тривалої й ефективної експлуатації платформи. Крім того, модульність архітектури дозволить легко масштабувати систему, адаптуючи її під нові вимоги ринку та технологічні зміни.

Таким чином, дотримання зазначених вимог у процесі проєктування та реалізації веб-застосунку є критично важливим для досягнення цілей розробки

інноваційної, гнучкої та конкурентоспроможної інтелектуальної системи прокату автомобілів.

На рисунку 2.1.1 представлено діаграму прецедентів веб-застосунку.

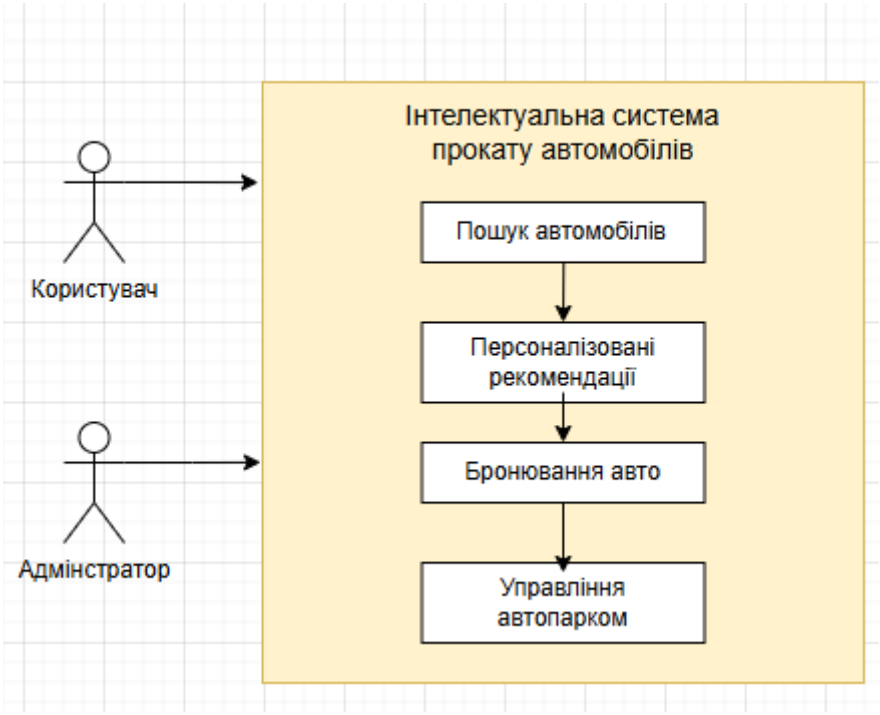


Рисунок 2.1.1 – Діаграма прецедентів (Use Case Diagram)

Діаграма прецедентів представляє узагальнене графічне зображення основних сценаріїв взаємодії користувача з інтелектуальною системою прокату автомобілів. Вона відображає послідовність дій, які виконує користувач під час роботи із системою, починаючи від пошуку автомобіля до завершення операції з бронювання.

У якості головних **акторів** виступають *Користувач* та *Адміністратор*, які взаємодіють із системою. Сценарії (прецеденти), що реалізуються у рамках цієї взаємодії, згруповані в межах прямокутної області, яка символізує систему.

Діаграма складається з таких основних прецедентів:

1. Пошук автомобілів: процес, під час якого користувач задає ключові параметри пошуку (бренд, модель, клас, тип пального, діапазон цін тощо) для перегляду доступних варіантів транспорту.
2. Персоналізовані рекомендації: інтелектуальний підпроцес, що активується після пошуку. Система аналізує вподобання користувача, історію замовлень, бюджет і контекстні фактори для формування оптимальних пропозицій автомобілів, найбільш відповідних запиту.
3. Бронювання авто: сценарій, який охоплює вибір автомобіля, зазначення строку прокату, оформлення додаткових послуг та підтвердження замовлення. Сюди також входить розрахунок остаточної вартості оренди та оплата.
4. Управління автопарком: адміністративний прецедент, який використовується лише для спеціальних ролей (наприклад, менеджера системи). Він включає додавання, редагування, деактивацію або видалення автомобілів із бази, а також оновлення їх характеристик та статусу доступності.

Загалом, діаграма демонструє логічну структуру основних дій у системі, чітко відокремлюючи обов’язки користувача від системних функцій.

Такий підхід забезпечує прозорість у визначенні вимог до функціоналу майбутнього веб-застосунку та спрощує подальше проєктування інтерфейсу та модулів системи.

2.2 Логіка функціонування інтелектуальної системи прокату автомобілів

Інтелектуальна система прокату автомобілів функціонує на основі послідовної обробки запитів користувача. Основна логіка побудована навколо взаємодії між клієнтом і системою рекомендацій, базою даних автомобілів, а також сервісами бронювання та оплати.

Після авторизації або входу користувач має змогу скористатися інтерфейсом для фільтрації доступного автотранспорту, отримання персоналізованих пропозицій, перегляду детальної інформації про авто, оформлення заявки та завершення процедури прокату.

На рисунку 2.2.1 представлено діаграму діяльності, яка ілюструє загальний алгоритм дій користувача у системі.

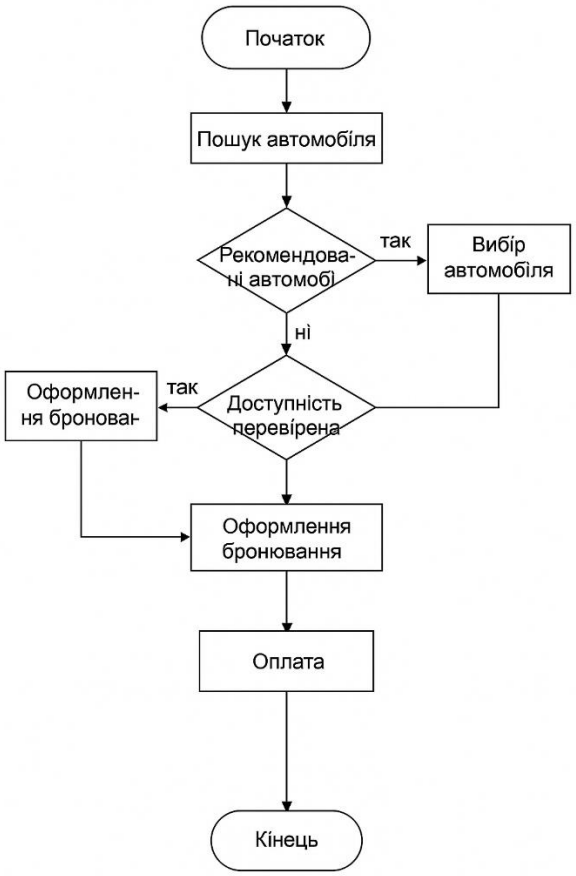


Рисунок 2.2.1 – Діаграма діяльності інтелектуальної системи прокату автомобілів

Основні етапи логіки функціонування системи:

1. Користувач відкриває веб-застосунок та проходить авторизацію (за потреби).
2. Ініціюється пошук авто за заданими критеріями.
3. Система аналізує параметри пошуку та історію користувача.

4. Виводиться список персоналізованих результатів.
5. Користувач обирає автомобіль і переходить до етапу бронювання.
6. Система перевіряє доступність авто, розраховує вартість, надає можливість вибрати додаткові послуги.
7. Після підтвердження даних система оформлює бронювання та (опціонально) проводить онлайн-оплату.
8. Адміністративна частина отримує оновлення про статус бронювання.
9. Уся інформація зберігається в базі для подальшого аналізу, рекомендацій і підтримки клієнта.

2.3 Проєктування інтерфейсу користувача

Інтерфейс користувача враховує вимоги до зручності, логічності та послідовності виконання дій. Основною метою є забезпечення інтуїтивного користувацького досвіду на всіх етапах взаємодії з веб-застосунком — від входу на сайт до остаточного підтвердження замовлення та перегляду результатів.

Проєктування інтерфейсу базується на принципах:

1. Модульності: кожний етап представлений окремою логічною сторінкою.
2. Адаптивності: інтерфейс зручно працює на ПК, планшетах і мобільних пристроях.
3. Мінімалізму та візуальної ієрархії: найважливіші дії виділяються, а другорядні залишаються непомітними;
4. UX-оптимізації: зменшення кількості натискань до мети, підказки, попередження про помилки, швидкий доступ до потрібної функції.

Для узагальненого уявлення про логіку побудови інтерфейсу було створено діаграму структури інтерфейсу (рис. 2.3.1), яка візуально демонструє основні етапи шляху користувача у системі, починаючи з головної сторінки й завершуючи фінальною конфігурацією замовлення.

На діаграмі зображено послідовність ключових елементів інтерфейсу:

1. Головна сторінка: перший екран, що привертає увагу користувача і надає змогу швидко перейти до початку взаємодії. Тут реалізовано меню навігації, промоційний блок та CTA-кнопки для пошуку або авторизації.
2. Вибір автомобіля: користувач переходить до сторінки, де доступна фільтрація автопарку за різними характеристиками: клас авто, ціна, тип пального, трансмісія, бренд тощо. Відображення реалізовано у вигляді карток із короткою інформацією.
3. Персоналізовані рекомендації: на основі введених параметрів, історії переглядів та поведінкових факторів формується список рекомендованих авто. Це дозволяє прискорити процес підбору автомобіля.
4. Перевірка доступності: при виборі конкретного авто система перевіряє його наявність на обрані дати. Якщо автомобіль недоступний, пропонується альтернатива з аналогічними характеристиками.
5. Бронювання: користувач вводить дані для підтвердження замовлення, обирає додаткові опції (страхування, навігатор, дитяче крісло), переглядає загальну вартість та натискає кнопку «Підтвердити». Можлива також оплата онлайн через інтегровані платіжні системи.
6. Результати: система надає підсумок операції, надсилає підтвердження на пошту або в особистий кабінет користувача.
7. Перегляд відгуків: користувач має можливість ознайомитися з оцінками інших клієнтів або залишити власний відгук про прокат, сервіс чи конкретний автомобіль.
8. Фінальна конфігурація/підтвердження: фінальна сторінка, яка дозволяє зберегти або роздрукувати деталі замовлення, а також повернутися до пошуку або повторити прокат.

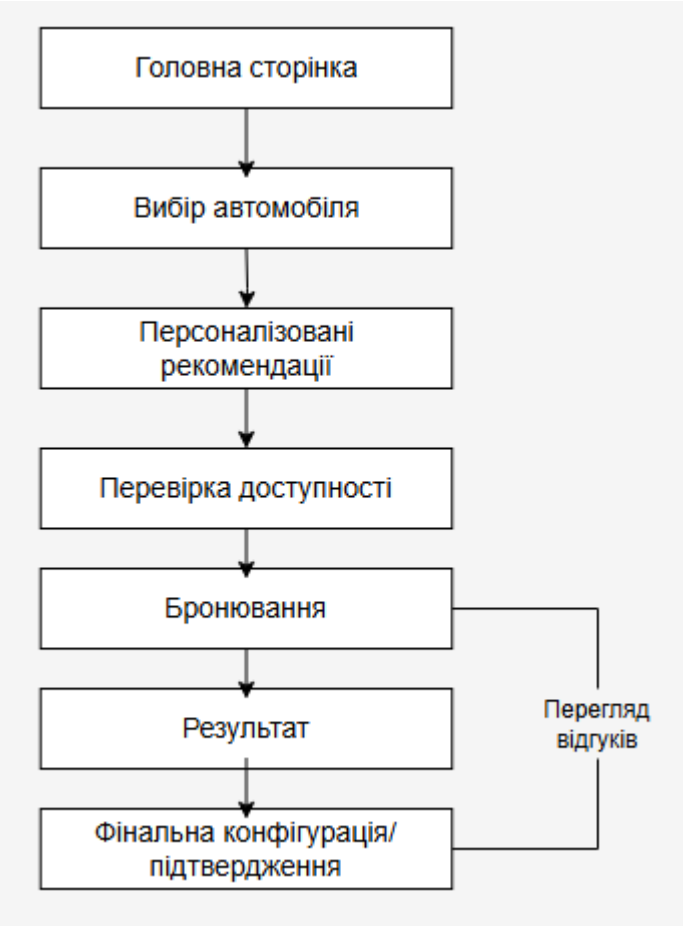


Рисунок 2.3.1 – Діаграма структури інтерфейсу (Interface Structure Diagram)

2.4 Моделювання процесів (опис алгоритмів машинного навчання)

У процесі розробки інтелектуальної системи ключовим завданням є забезпечення релевантного, персоналізованого підбору транспортного засобу відповідно до запиту користувача. З цією метою було обрано низку алгоритмів машинного навчання, які забезпечують ефективну обробку вхідних даних, генерацію рекомендацій та оптимізацію вибору з урахуванням параметрів автопарку.

Алгоритм 1 Найближчих сусідів (*k*-Nearest Neighbors, k-NN)

Даний алгоритм є одним із найпоширеніших методів класифікації та регресії, що базується на принципі подібності [7]. У контексті рекомендаційної

системи прокату авто, *k*-NN дозволяє знаходити *k* найближчих варіантів транспортних засобів у багатовимірному просторі ознак, враховуючи такі параметри, як:

1) бюджет користувача,
2) тип палива,
3) трансмісія,
4) категорія авто,
5) клас комфортності тощо.

Відстань між об’єктами вимірюється за допомогою евклідової метрики або косинусної подібності, залежно від попередньої нормалізації даних. На основі найближчих записів формуються рекомендації, які найбільш відповідають запиту користувача.

Переваги алгоритму:

1. Простота реалізації та мінімальні вимоги до налаштування.
2. Висока точність при наявності добре структурованих даних.
3. Інтуїтивно зрозуміла логіка результатів.

Недоліки алгоритму:

1. Високі обчислювальні витрати при великому обсязі даних.
2. Необхідність попередньої нормалізації ознак.
3. Чутливість до шуму в даних і вибору значення *k*.

Алгоритм 2 Кластеризація методом *k*-середніх (*k*-Means)

Метод *k*-середніх застосовано на етапі сегментації автопарку. Його основна ідея полягає у розбитті множини автомобілів на *k* кластерів, у межах яких транспортні засоби мають схожі технічні характеристики (потужність, тип кузова, вартість, витрата пального тощо) [8]. Такий підхід дозволяє оптимізувати процес пошуку, звузивши його до одного релевантного кластера.

Алгоритм ітеративно обчислює центри кластерів (центроїди) і перевизначає об’єкти, поки розподіл не стабілізується.

Переваги алгоритму:

1. Швидка збіжність при великій кількості даних.

2. Можливість попередньої агрегації даних для подальшої оптимізації.
3. Простота реалізації та візуалізації.

Недоліки алгоритму:

1. Схильність до локальних мінімумів.
2. Погана ефективність при кластерах нерегулярної форми.
3. Вразливість до вибору початкових центрів.

Алгоритм 3 Дерева рішень (Decision Trees)

Для реалізації пояснюваної логіки вибору транспортного засобу було використано алгоритм дерева рішень, який дозволяє формувати набір умовних переходів на основі заданих критеріїв (наприклад, бюджет < 80$, клас = "компакт", трансмісія = "автомат"). Цей підхід дозволяє реалізувати логіку підбору у вигляді набору "Якщо – Тоді", що полегшує інтерпретацію як для розробника, так і для кінцевого користувача [9].

Переваги алгоритму:

1. Зрозуміла логіка побудови рішень.
2. Підтримка категоріальних та числових змінних.
3. Можливість графічного представлення процесу вибору.

Недоліки алгоритму:

1. Схильність до перенавчання.
2. Потреба у додатковій обрізці дерева або ансамблевих підходах (Random Forest, Gradient Boosting).

Алгоритм 4 Модель GPT

З огляду на сучасні тенденції інтеграції мовних моделей, у проєкті розглядається можливість застосування великої трансформерної моделі, такої як GPT-4o, для обробки неструктурованого запиту користувача у текстовій формі [17].

Це дозволить:

1. Аналізувати вільний текст (наприклад, "Шукаю спортивне авто з низьким пробігом до 150$ на день").
2. Витягувати ключові характеристики із запиту.

3. Здійснювати семантичний пошук за embedding-вектором.

Переваги моделі:

1. Гнучкість і контекстність при формулюванні запиту.
2. Потенціал обробки неструктурованої інформації (відгуки, коментарі).
3. Можливість інтеграції в розділ пошуку або чату з користувачем.

Недоліки моделі:

1. Необхідність зовнішніх API-запитів (OpenAI).
2. Вартість та залежність від продуктивності моделі.
3. Потреба в додатковому контролі за відповідністю даних політиці безпеки.

Таким чином, підбір моделей машинного навчання здійснювався з урахуванням наступних чинників:

1. Обсяг і структура даних: обмежена кількість записів, різні типи ознак (числові, категоріальні).
2. Інтерпретованість результатів: необхідність пояснити користувачу логіку підбору (дерева рішень).
3. Масштабованість: можливість розширення системи при зростанні обсягів даних та автопарку (кластеризація).
4. Гнучкість: підтримка адаптації до нових запитів, параметрів та впровадження AI-компонентів (GPT).

Такий набір алгоритмів забезпечує не лише високу адаптивність, а й надійну роботу системи у реальних умовах функціонування прокату автомобілів.

2.5 Моделювання даних (опис структури бази даних)

У межах проєктування інтелектуального веб-застосунку для прокату автомобілів важливу роль відіграє належно побудована структура бази даних. База даних має забезпечувати надійне зберігання інформації про транспортні засоби, користувачів, оренди, технічні параметри, ціни, а також підтримку

функцій машинного навчання, які використовуються для реалізації рекомендаційної логіки.

Для зберігання даних обрано реляційну модель, реалізовану на основі PostgreSQL у середовищі Supabase. Базу даних побудовано з урахуванням принципів нормалізації, масштабованості та можливості подальшого розширення.

Система містить п’ять основних таблиць: Users, Cars, Rentals, Feedback та Recommendations. Вони утворюють логічно зв’язану структуру, яка забезпечує персоналізований підбір авто, зберігання історії бронювань, облік відгуків і функціонування механізму рекомендацій.

Таблиця Users (табл. 2.5.1) є фундаментом усієї системи, оскільки зберігає облікові дані зареєстрованих користувачів. Відповідає за ідентифікацію клієнтів, персоналізацію рекомендацій, формування історії оренд та управління взаємодією з інтерфейсом користувача.

Призначення:

1) автентифікація та авторизація користувачів;
2) основа для побудови рекомендаційних моделей (наприклад, врахування стажу водіння);
3) зв’язок із транзакційною історією (оренда, відгуки, рекомендації).

Таблиця 2.5.1 – Структур таблиці Users

№	Поле	Тип	Призначення
1	UserID	UID	Унікальний ідентифікатор користувача(PK)
2	Email	text	Електронна пошта для автентифікації
3	first_name	text	Ім’я користувача
4	last_name	text	Прізвище користувача
5	city	text	Місто проживання
6	country	text	Країна проживання
7	phone	text	Контактний номер телефону
8	driving experience	text	Стаж водіння

За допомогою зовнішнього ключа має зв’язок з таблицями Rentals, Feedback, Recommendations.

Таблиця Cars (табл. 2.5.2) є ключовим джерелом даних про всі транспортні засоби, що доступні для оренди у системі. Містить технічні та економічні характеристики кожного авто, а також візуальну інформацію, необхідну для відображення в інтерфейсі.

Призначення:

1) забезпечення відбору автомобілів відповідно до фільтрів користувача;
2) підтримка роботи системи рекомендацій, кластеризації та машинного навчання;
3) відображення детальної інформації про авто на сторінці вибору.

Таблиця 2.5.2 – Структура таблиці Cars

№	Поле	Тип	Призначення
1	CarID	int	Первинний ключ. Унікальний ідентифікатор авто
2	category	text	Клас авто
3	brand	text	Марка автомобіля
4	model	text	Модель авто
5	generation	text	Покоління або серія випуску
6	year	int	Рік випуску
7	fuel	text	Тип палива
8	transmisson	text	Тип трансмісії
9	drive	text	Тип приводу
10	engine	text	Характеристика двигуна
11	horsepower	int	Потужність двигуна
12	torque	int	Крутий момент
13	consumptionCity	float	Витрата пального в міському циклі
14	consumptionHighway	float	Витрата пального на трасі
15	price_day	float	Ціна оренди за добу
16	price_week	float	Ціна оренди за тиждень (доба)
17	price_month	float	Ціна оренда за місяць (доба)
18	description	text	Текстовий опис автомобіля
19	Image_url	text	Посилання на зображення авто

За допомогою зовнішнього ключа має зв’язок з таблицями Rentals, Feedback, Recommendations.

Таблиця Rentals (табл. 2.5.3) є транзакційною сутністю, яка фіксує кожний факт прокату автомобіля конкретним користувачем. Містить часову інформацію, а також суму до сплати, яка може бути обчислена на основі тарифів із таблиці Cars.

Призначення:

1) відображення історії оренди;
2) база для формування відгуків;
3) джерело інформації для аналітики й побудови моделей користувацької поведінки.

Таблиця 2.5.3 – Структура таблиці Rentals

№	Поле	Тип	Призначення
1	RentalID	UID	Унікальний ідентифікатор бронювання (PK)
2	UserID	UID	Зовнішній ключ - Users
3	CarID	int	Зовнішній ключ - Cars
4	StartDate	date	Дата початку оренди
5	EndDate	date	Дата завершення оренди
6	TotalPrice	numeric	Підсумкова вартість оренди

За допомогою зовнішнього ключа напряму пов’язана з таблицями Users та Cars, дозволяє відстежити, хто і коли орендував автомобіль.

Таблиця Feedback (табл. 2.5.4) служить інструментом зворотного зв’язку. Дозволяє оцінити якість автомобіля з боку користувача та зберігає числову оцінку та текстовий коментар.

Дані можуть бути використані для побудови репутаційної системи авто та підвищення якості обслуговування.

Таблиця 2.5.4 – Структура таблиці Feedback

№	Поле	Тип	Призначення
1	UserID	UID	Зовнішній ключ – Users (частина РК)
2	CarID	int	Зовнішній ключ – Cars (частина РК)
3	Rating	int	Оцінка авто від користувача
4	Comment	float	Текстовий коментар від користувача

Призначення:

1) забезпечення прозорості сервісу через відкриті відгуки;
2) вплив на рекомендаційну систему через оцінки;
3) основа для модерації або ранжування авто.

За допомогою зовнішнього ключа має зв’язок з таблицями Users та Cars.

Таблиця Recommendations (табл. 2.5.5) є основою рекомендаційного модуля. Дозволяє зафіксувати, які авто система порадила конкретному користувачеві, наскільки вони релевантні та чому саме ці авто були обрані.

Таблиця також може слугувати журналом для тестування і порівняння різних алгоритмів підбору.

Призначення:

1) зберігання результатів ML/AI-рекомендацій;
2) пояснення вибору (наприклад, відповідність бюджету, тип палива);
3) основа для A/B тестування якості рекомендацій.

Таблиця 2.5.5 – Структура таблиці Recommendations

№	Поле	Тип	Призначення
1	UserID	UID	Зовнішній ключ – Users (частина РК)
2	CarID	int	Зовнішній ключ – Cars (частина РК)
3	Score	float	Оцінка релевантності рекомендації
4	Reasons	text	Пояснення причини вибору

За допомогою зовнішнього ключа має зв’язок з таблицями Users та Cars що, дозволяє сформувати історію взаємодії користувача з системою рекомендацій.

На рисунку 2.5.1 представлено діаграму «сутність-зв’язок» (Entity Relationship Diagram), яка наочно представляє структуру даних інтелектуальної системи та взаємозв’язків між ними.

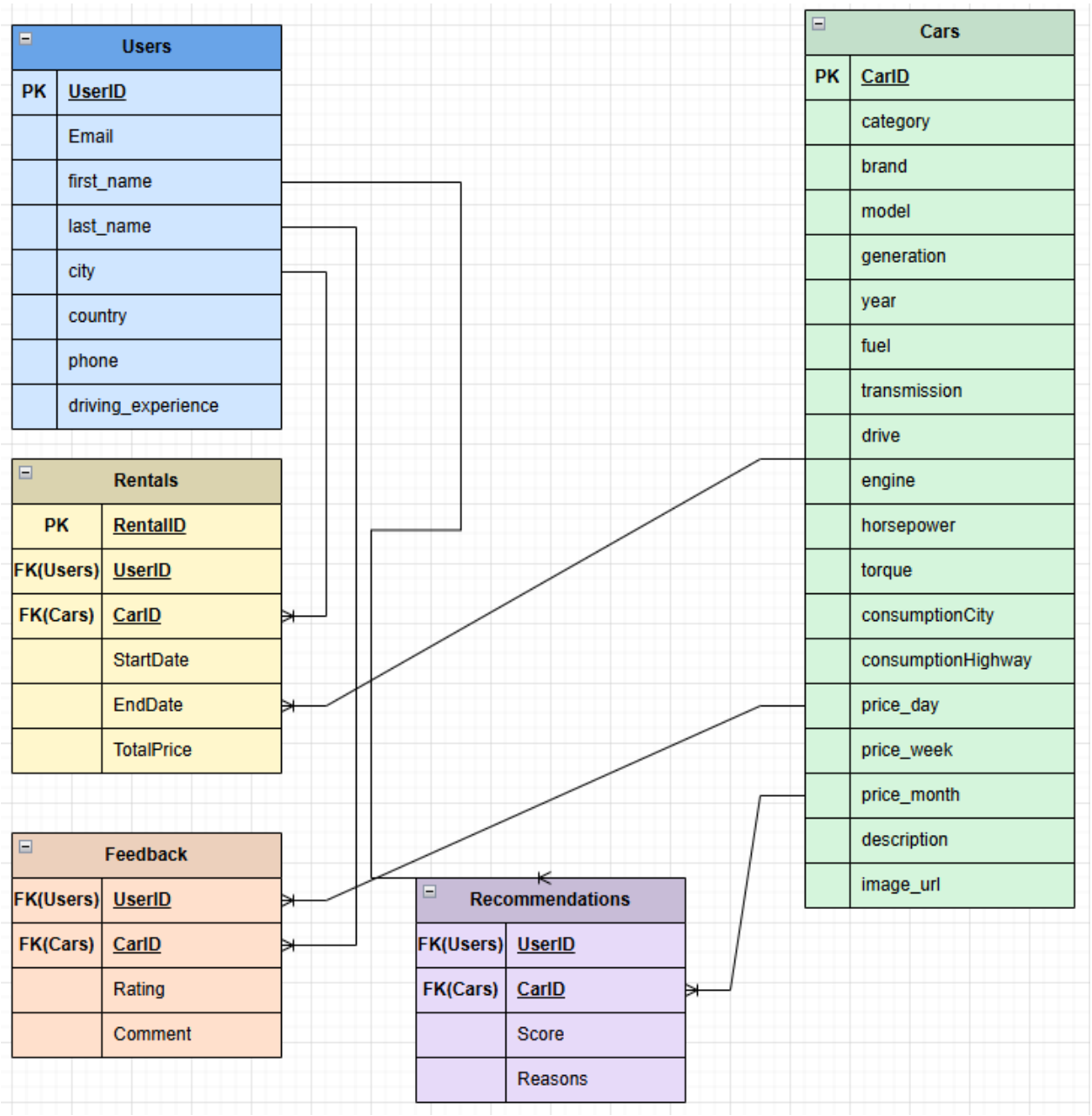


Рисунок 2.5.1 – ER-діаграма бази даних

Спроєктована база даних відповідає вимогам до сучасних веб-застосунків, оскільки є логічно структурованою, оптимізованою під аналітику та підтримує масштабування. Згідно з концепцією системи, дані не лише обслуговують користувацький інтерфейс, а й активно залучаються до побудови інтелектуальних рекомендацій. Передбачена можливість додавання нових сутностей дозволяє інтегрувати додатковий функціонал без порушення існуючої структури.

Висновки до розділу 2

У другому розділі розглянуто ключові аспекти проєктування інтелектуального веб-застосунку для прокату автомобілів. Сформульовано функціональні та нефункціональні вимоги до системи, що враховують специфіку використання алгоритмів персоналізованих рекомендацій, онлайн-бронювання, управління автопарком та взаємодії з користувачем.

Спроєктовано логіку функціонування системи відповідно до принципів модульності та послідовності дій користувача. Побудовано діаграму діяльності, яка описує основні етапи взаємодії із системою – від ініціації пошуку до завершення бронювання. Окремо розглянуто концепцію інтерфейсу користувача, орієнтовану на сучасний візуальний стиль та простоту навігації.

Проведено моделювання процесів рекомендації авто з використанням алгоритмів машинного навчання. Розглянуто та обґрунтовано вибір таких підходів: алгоритм найближчих сусідів (k-NN), кластеризація методом k-середніх (k-Means), дерева рішень (Decision Trees), а також потенційна інтеграція великої мовної моделі GPT.

Описано структуру реляційної бази даних на основі PostgreSQL у Supabase. Структуру даних адаптовано для підтримки роботи рекомендаторів і подальшої взаємодії з модулями бронювання та AI.

Таким чином, у розділі узагальнено архітектурні, логічні, інтерфейсні та аналітичні рішення, що формують основу для реалізації інтелектуальної, масштабованої та зручної системи прокату автомобілів, здатної задовольнити потреби сучасного цифрового користувача.

3 РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ

3.1 Вибір технологій для розробки

Розробка інтелектуального веб-застосунку прокату автомобілів вимагала ретельного вибору інструментів, які б відповідали низці технічних і функціональних вимог. До таких вимог належать: підтримка сучасного інтерфейсу користувача, інтеграція з модулями машинного навчання, доступ до бази даних у реальному часі, безпечна авторизація, швидкий пошук і адаптивність до мобільних пристроїв.

Під час аналізу технологічного стеку було враховано критерії:

1. Швидкість розробки MVP та простота впровадження.
2. Можливість масштабування.
3. Гнучкість роботи з API та сторонніми сервісами (зокрема OpenAI).
4. Наявність спільноти, документації та підтримки.
5. Оптимізація продуктивності на стороні клієнта і сервера.
6. Висока інтерактивність і зручність для користувача.

React.js обрано як базову технологію для створення інтерфейсу користувача. Ця бібліотека надає потужні інструменти для побудови компонентної архітектури, повторного використання елементів UI, керування станом застосунку та динамічного оновлення вмісту сторінок без повного перезавантаження [10]. Проаналізовано альтернативи Angular або Vue.js, проте React:

1) має найширшу екосистему компонентів та бібліотек;
2) найкраще інтегрується з Next.js (який також використано в цьому проєкті);
3) є більш гнучким у проєктуванні структури застосунку.

Next.js обрано у якості фреймворку для серверного рендерингу, маршрутизації, оптимізації сторінок і побудови API-ендпоінтів [12]. Він дозволяє:

1) створювати динамічні сторінки із завантаженням даних на сервері;
2) покращувати SEO за рахунок server-side rendering;
3) уникати створення окремого бекенду – завдяки API-роутам (наприклад, для інтеграції з GPT або Supabase);
4) реалізувати легкий деплой через Vercel без складної інфраструктури.

На відміну від звичайного React SPA (Single Page Application), Next.js дозволив реалізувати гібридну архітектуру: частина сторінок є статичними, частина – генерується динамічно, що зменшує навантаження на клієнт.

Supabase обрано у якості хмарного бекенду [14]. Це open-source платформа, що поєднує реляційну базу даних PostgreSQL, модулі авторизації та зберігання файлів. Supabase забезпечує:

1) швидке створення таблиць та майбутніх bookings;
2) автоматичну генерацію RESTful API для доступу до таблиць;
3) зручне управління авторизацією, автентифікацією користувачів через email;
4) підтримку Row Level Security (RLS) – політика доступу до даних;
5) хостинг зображень авто через модуль Supabase Storage.

Розглянуто альтернативи Firebase, MongoDB Atlas, Hasura, проте вибір Supabase обґрунтований:

1) кращою структурованістю реляційних даних (що підходить для фільтрації авто за параметрами);
2) SQL-сумісністю;
3) меншою складністю у налаштуванні політик безпеки.

Tailwind CSS обрано для стилізації інтерфейсу [13]. Це утилітарний CSS-фреймворк, який дозволяє писати стилі безпосередньо в JSX-компонентах за допомогою класів. Порівняно з Bootstrap або CSS-in-JS (Emotion, Styled Components), Tailwind забезпечує:

1) високу швидкість створення адаптивного дизайну;
2) кращий контроль над інтерфейсом;
3) менший обсяг фінального CSS-коду (завдяки purge-механізму).

Tailwind особливо ефективний у проєктах з великою кількістю UI-компонентів та анімацій. Саме такий підхід реалізовано у цьому веб-застосунку.

TypeScript обрано для покращення підтримки коду та уникнення помилок під час розробки, як надбудову над JavaScript [11]. Його використання дозволило:

1) типізувати пропси компонентів;
2) явно описувати структури об’єктів (наприклад, Car, User);
3) пришвидшити відлагодження;
4) покращити читабельність і масштабованість проєкту.

GPT-4o (OpenAI API) Для впровадження інтелектуальних функцій у системі передбачено інтеграцію з GPT-4o через API від OpenAI [17]. Модель дозволяє:

1) обробляти текстові запити користувача у довільній;
2) семантично інтерпретувати наміри і зіставляти їх з наявними авто;
3) реалізувати AI-асистента у формі чат-інтерфейсу або анкетного підбору.

Використання GPT-4o обрано завдяки його багатофункціональності (обробка тексту, коду, зображень), широкій підтримці, стабільності, а також зручному механізму інтеграції через REST API.

Додаткові технології

1. Framer Motion: бібліотека для створення анімацій у React.
2. Vercel: платформа для хостингу та CI/CD Next.js застосунків.
3. Git + GitHub: система контролю версій, спільна розробка, history tracking.
4. Lucide / Heroicons: SVG-іконки для сучасного візуального оформлення.

3.2 Реалізація веб-застосунку

Реалізація веб-застосунку складається з побудови функціональних модулів, що поділяються на клієнтську частину (frontend), серверну логіку

(API-ендпоінти), зв’язок із базою даних, інтеграцію з моделлю штучного інтелекту та обробку запитів користувача. Компонентна архітектура дозволила забезпечити гнучкість, повторне використання логіки, а також адаптивність інтерфейсу.

Для зберігання даних про користувачів, автомобілі та запити використовується реляційна база даних **PostgreSQL**, що розгорнута у хмарному середовищі **Supabase**. Підключення до бази даних здійснюється за допомогою клієнта Supabase, ініціалізованого у файлі supabaseClient.ts.

На рисунку 3.2.1 представлено ініціалізацію клієнта Supabase через метод createClient, що імпортується з бібліотеки @supabase/supabase-js. З’єднання з базою відбувається на основі змінних середовища NEXT_PUBLIC_SUPABASE_URL, NEXT_PUBLIC_SUPABASE_ANON_KEY, які зберігають URL до інстанції Supabase та публічний ключ доступу відповідно. Ініціалізований клієнт supabase надалі використовується в усіх частинах проєкту для читання та запису даних у таблиці бази даних.

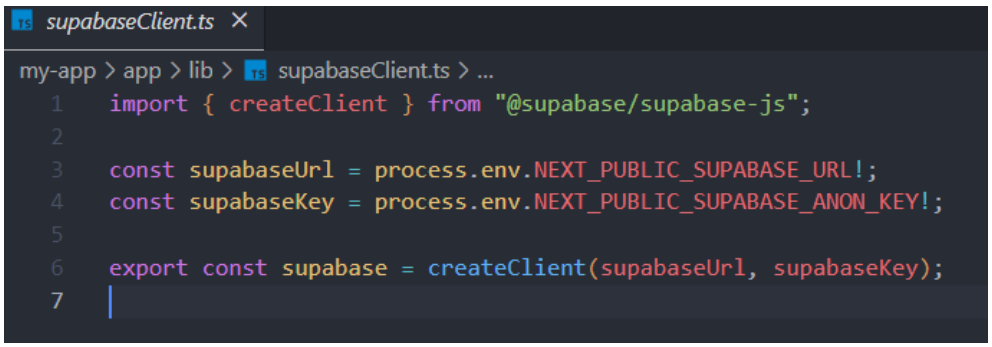


Рисунок 3.2.1 – Код ініціалізації клієнта Supabase

На сторінці пошуку автомобілів реалізовано логіку завантаження списку транспортних засобів із таблиці БД. Для цього використовується клієнт Supabase. У React-компоненті CarSearchPage визначено хуки стану для зберігання списку автомобілів та індикатора завантаження. Запит до бази

даних виконується у функції fetchCars, яка викликається всередині useEffect() при монтуванні компонента.

При завантаженні компонента здійснюється асинхронний запит supabase.from("cars").select(), який повертає всі записи з таблиці автомобілів. Дані зберігаються у локальному стані за допомогою хуків useState. У разі виникнення помилки вона виводиться до консолі. Після завершення запиту індикатор isLoading вимикається, а результати передаються у відповідний компонент вітрини (CarList), де формуються картки автомобілів (рис. 3.2.2).

Рисунок 3.2.2 – Код сторінки пошуку авто

Дані про автомобілі запитуються з таблиці cars. Реалізовано динамічну фільтрацію за параметрами: марка, модель, клас, тип палива, привід, ціна. Користувач обирає фільтри через форму, а результати оновлюються в режимі реального часу.

На рисунку 3.2.3 представлено реалізацію компонента CarFilter, який відповідає за формування інтерфейсу фільтрації та логіку фільтрації масиву автомобілів.

Компонент приймає список cars як пропс, а також функцію onFilter для передачі відфільтрованих результатів у батьківський компонент.

У середині компонента створено локальні стани для зберігання значень фільтрів (selectedBrand, selectedFuel, priceRange тощо), а також функцію resetFilters() для очищення всіх обраних параметрів.

Метод useEffect() спрацьовує кожного разу, коли змінюється будь-яке з фільтрувальних значень, і виконує фільтрацію списку автомобілів відповідно до вибраних критеріїв. Результат передається через onFilter(filtered) у компонент CarList. Додатково функція unique() дозволяє динамічно формувати список опцій для кожного фільтру. Це забезпечує узгодженість між доступними параметрами та даними, наявними у базі.

Рисунок 3.2.3 – Реалізація компонента CarFilter

На рисунку 3.2.4 представлено реалізацію компонента CarList, який відповідає за відображення результатів фільтрації автомобілів. Цей компонент приймає масив cars у вигляді пропсу та виводить відповідні картки авто за допомогою дочірнього компонента CarCard.

Якщо масив cars порожній, на екран виводиться повідомлення *"No cars match your filters."*, що інформує користувача про відсутність результатів згідно з обраними критеріями. У протилежному випадку відбувається ітерація по списку автомобілів з формуванням візуального представлення кожної моделі.

На сторінці з деталями відображається розширена інформація про автомобіль, включаючи технічні характеристики, витрати пального, ціни оренди, генерацію на основі параметра id у URL.

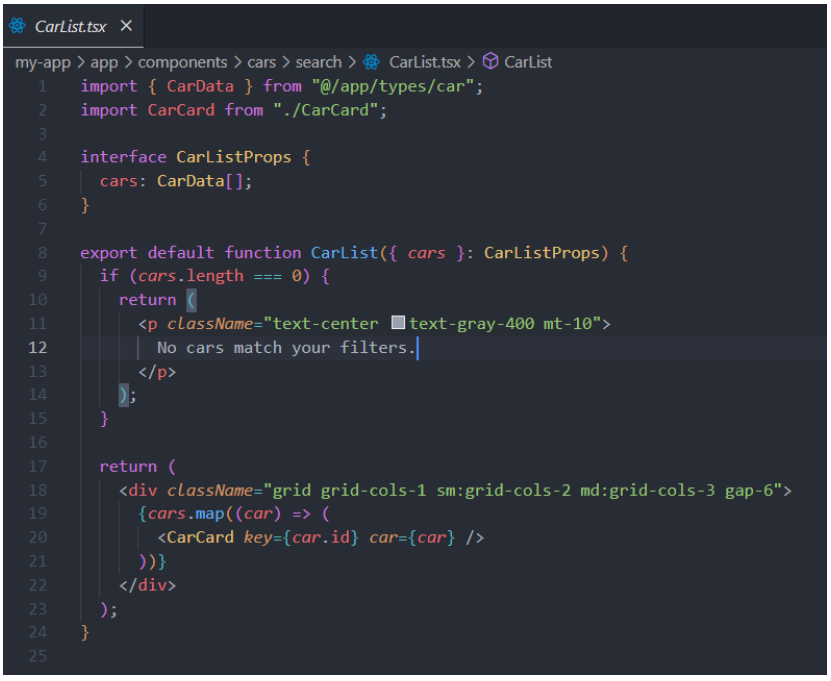


Рисунок 3.2.4 – Реалізація компонента CarList

На рисунку 3.2.5 представлено фрагмент реалізації компонента CarDetailPage, що відповідає за отримання даних про конкретний автомобіль із бази даних Supabase. Ідентифікатор авто зчитується з параметра params.id,

після чого здійснюється запит до таблиці cars методом .select().eq("id", carId).single().

Якщо запит не повертає результат або виникає помилка, сторінка перенаправляє користувача на сторінку з повідомленням notFound(). У разі успішного запиту інформація про авто передається до підкомпонентів для відображення технічних характеристик (Performance, VehicleInfo, FuelConsumption, RentalPrices) та блоку бронювання (BookCarSection).

Такий підхід дозволяє модульно компонувати інтерфейс сторінки та забезпечити відображення повної специфікації транспортного засобу.

```
import { supabase } from "@/app/lib/supabaseClient";
import { notFound } from "next/navigation";
import Image from "next/image";
import PageWrapper from "@/app/components/pageWrapper/PageWrapper";

import Performance from "@/app/components/cars/id/Perfomance";
import RentalPrices from "@/app/components/cars/id/RentalPrices";
import VehicleInfo from "@/app/components/cars/id/VehicleInfo";
import FuelConsumption from "@/app/components/cars/id/FuelConsumption
import BookCarSection from "@/app/components/cars/id/BookCarSection";

export default async function CarDetailPage({
  params,
}: {
  params: Promise<{ id: string }>;
}) {
  const resolved = await params;
  const carId = parseInt(resolved.id, 10);
  const { data: car, error } = await supabase
    .from("cars")
    .select("*")
    .eq("id", carId)
    .single();

  if (error || !car) return notFound();

  return (···
  );
}
```

Рисунок 3.2.5 – Реалізація компонента CarDetailPage

На рисунку 3.2.6 представлено частину JSX-розмітки компонента CarDetailPage, яка відповідає за відображення візуального представлення автомобіля. Зображення авто передається у вигляді URL (car.image_url) та виводиться через компонент Image із застосуванням стилів Tailwind CSS для

адаптивності та візуальної привабливості (object-cover, rounded-xl, shadow-2xl тощо).

Нижче реалізовано сіткову структуру (grid grid-cols-1 lg:grid-cols-2) для відображення додаткових характеристик автомобіля: ціна за місяць (price_month), тиждень (price_week), добу (price_day). Значення передаються до дочірнього компонента RentalPrices, який форматовано відповідним чином для покращення сприйняття користувачем.

Крім того, на сторінці присутні компоненти CarDescription (для виведення текстового опису) та BookCarSection – модуль, відповідальний за інтеграцію з функціоналом бронювання. Це забезпечує комплексне представлення інформації про автомобіль та підготовку до подальших дій користувача.

```
return (
  <PageWrapper>
    <div className="max-w-6xl mx-auto py-12 px-6 animate-fade-in">
      <h1 className="text-4xl font-black text-center mb-6">
        {car.brand} {car.model}{" "}
        <span className="text-gray-500 font-semibold text-2xl">···
        </span>
      </h1>

      <div className="overflow-hidden rounded-3xl shadow-2xl transition-transform duration-500 hover:scale-[1.01]">
        <Image
          src={car.image_url || "/fallback.jpg"}
          alt={`${car.brand} ${car.model}`}
          width={1200}
          height={500}
          className="w-full h-[500px] object-cover rounded-3xl"
          priority
        />
      </div>

      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 mt-10">
        <RentalPrices
          price={car.price_month}
          pricePerDay={car.price_day}
          pricePerWeek={car.price_week}
        />
        <div className="flex flex-col gap-6">···
        </div>
      </div>

      {car.description && (···
      )}

      <BookCarSection />
    </div>
```

Рисунок 3.2.6 – Реалізація компонента CarDetailPage (продовження)

Реалізація збереження та оновлення профілю користувача представлено на рисунках 3.2.7 та 3.2.8. Після входу користувач може оновити свої дані (ім’я, місто, країна, телефон, досвід). Ці зміни зберігаються в таблиці users з автоматичною перевіркою авторизації.

Показано реалізацію логіки оновлення профілю користувача у компоненті ProfilePage. Після автентифікації користувач має змогу переглянути або змінити такі дані: ім’я, прізвище, місто, країна, номер телефону та досвід водіння.

За допомогою методу supabase.auth.getUser() отримується поточний авторизований користувач. Якщо такий користувач не знайдений, його перенаправляють на сторінку входу. Далі система намагається знайти відповідний профіль у таблиці users. Якщо запис існує – поля форми заповнюються отриманими даними; якщо ні – створюється новий запис (рис. 3.2.7).

На рисунку 3.2.8 показано механізм обробки змін у формі (handleChange) та збереження нових даних (handleSave). Використовується метод upsert – вставка нового запису або оновлення існуючого залежно від поля email. Усі значення з полів форми передаються до таблиці users з перевіркою на наявність конфлікту по унікальному email.

Рисунок 3.2.7 – Реалізація компонента ProfilePage

```
  useEffect(() => {
    const getUser = async () => {
                country: "",
                phone: "",
                driving_experience: "",
              },
            ]);
        }

        setLoading(false);
      };

      getUser();
    }, [router]);

    const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
      const { name, value } = e.target;
      setForm((prev) => ({ ...prev, [name]: value }));
    };

    const handleSave = async () => {
      if (!userEmail) return;
      setSaving(true);
      const { error } = await supabase.from("users").upsert(
        {
          email: userEmail,
          first_name: form.firstName,
          last_name: form.lastName,
          city: form.city,
          country: form.country,
          phone: form.phone,
          driving_experience: form.drivingExperience,
        },
        { onConflict: "email" }
      );
```

Рисунок 3.2.8 – Реалізація компонента ProfilePage (продовження)

Таким чином реалізовано зручний інтерфейс для редагування профілю з двосторонньою синхронізацією між формою та базою даних.

Для реалізації пошуку за текстовим запитом користувача використано модель GPT-4o через OpenAI API. Користувач вводить довільний запит, система надсилає його до API-ендпоінту /api/search-ai, де:

1. Генерується запит до GPT.
2. Отримується релевантна відповідь (наприклад, категорія або бренд).
3. Відбувається запит до таблиці cars для формування рекомендацій.

На рисунку 3.2.9 представлено фрагмент API-ендпоінту (route.ts), що обробляє HTTP-запит до /api/search-ai. У показаному прикладі реалізовано GET-запит до таблиці cars, який виконується за допомогою методу .select("*")

з клієнта Supabase. У разі помилки відповідь формується у вигляді JSON з повідомленням про невдачу та статусом 500.

Цей код є частиною більшого AI-модуля, в якому спочатку обробляється текстовий запит користувача за допомогою GPT-4o (наприклад, витягується релевантна категорія чи бренд). Далі результат передається у цю функцію, що повертає перелік автомобілів, які відповідають виявленим критеріям.

Важливою особливістю є те, що компонент використовує серверні API-роути, вбудовані у Next.js, що дозволяє уникнути створення окремого бекенду. Таким чином досягається спрощення архітектури застосунку при збереженні можливостей інтеграції зі сторонніми сервісами, зокрема OpenAI.

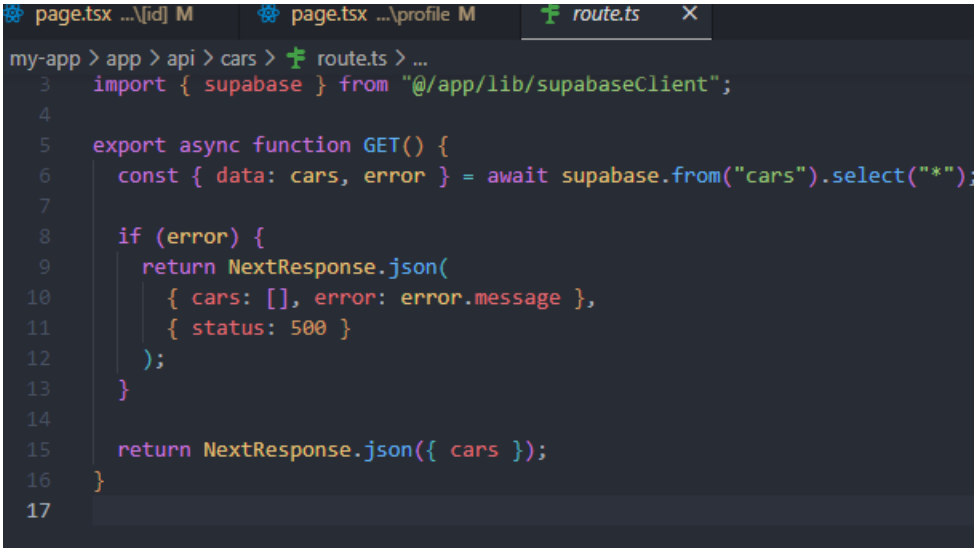


Рисунок 3.2.9 – Реалізація фрагменту API-ендпоінту

На рисунках 3.2.10–3.2.12 представлено повну реалізацію логіки обробки текстового запиту користувача для інтелектуального пошуку автомобілів на основі GPT-4o.

Рисунок 3.2.10 демонструє початкову частину API-ендпоінту /api/search-ai, де виконується парсинг тіла запиту (userInput), а також попереднє отримання мінімального набору даних про автомобілі з бази даних Supabase (id, brand, model). Цей список надалі використовується як джерело

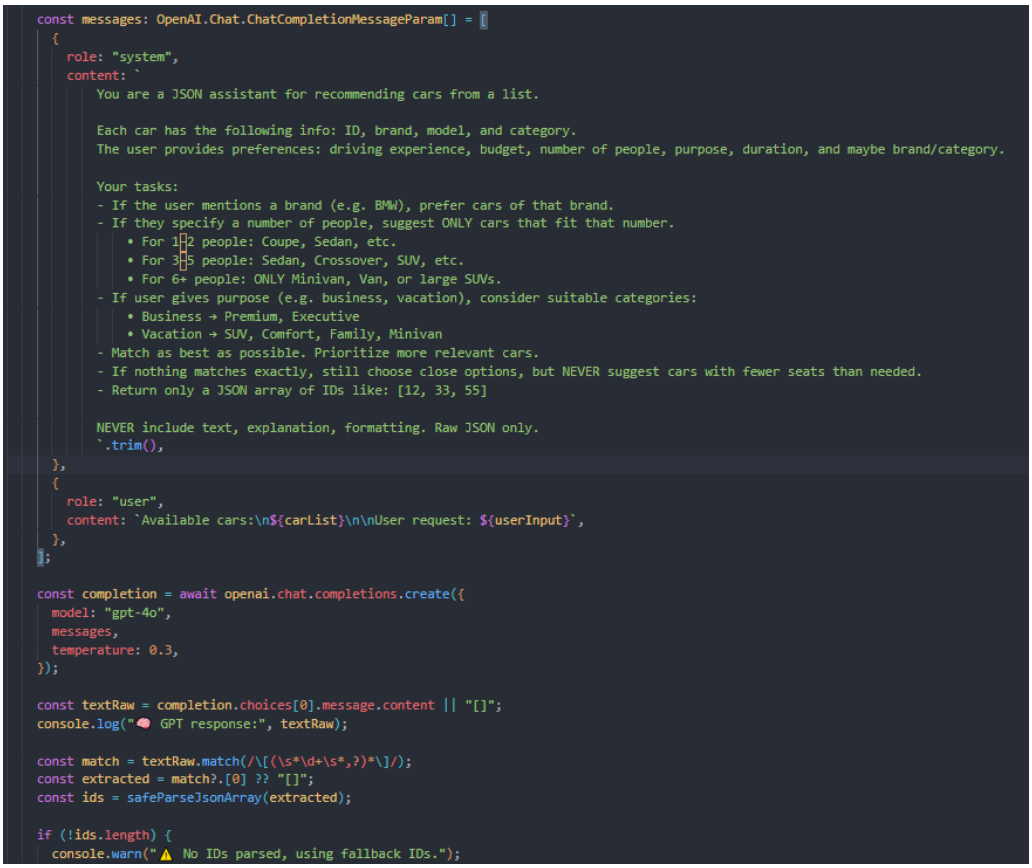
для аналізу та вибору GPT-моделлю. У разі відсутності даних користувач отримує відповідь зі статусом 500.

```
import { NextResponse } from "next/server";

const openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY! });

export async function POST(req: Request) {
  console.log("✅ Reached /api/search-ai");

  try {
    const { userInput } = await req.json();
    console.log("📥 User input:", userInput);

    const { data: cars, error } = await supabase
      .from("cars")
      .select("id, brand, model");

    if (error || !cars?.length) {
      console.error("❌ Supabase error:", error);
      return NextResponse.json(
        { reply: [], error: "No cars found." },
        { status: 500 }
      );
    }

    const carList = cars
      .map((car) => `ID: ${car.id}, ${car.brand} ${car.model}`)
      .join("\n");

    console.log("🚗 Car list sent to GPT:", carList);
```

Рисунок 3.2.10 – Реалізація початкової частини API-ендпоінту

Рисунок 3.2.11 демонструє формування prompt-запиту для моделі GPT-4o. Від імені системи (role: system) задаються правила генерації: модель повинна вибрати автомобілі, які відповідають введеним критеріям (досвід, бюджет, ціль, кількість пасажирів тощо), при цьому уникати зайвого тексту та повертати результат у форматі JSON-масиву ID.

Після формування запиту виконується виклик до openai.chat.completions.create, результат якого парситься через регулярний вираз і перетворюється на масив ID.



```
const messages: OpenAI.Chat.ChatCompletionMessageParam[] = [
  {
    role: "system",
    content: `
        You are a JSON assistant for recommending cars from a list.

        Each car has the following info: ID, brand, model, and category.
        The user provides preferences: driving experience, budget, number of people, purpose, duration, and maybe brand/category.

        Your tasks:
        - If the user mentions a brand (e.g. BMW), prefer cars of that brand.
        - If they specify a number of people, suggest ONLY cars that fit that number.
            • For 1–2 people: Coupe, Sedan, etc.
            • For 3–5 people: Sedan, Crossover, SUV, etc.
            • For 6+ people: ONLY Minivan, Van, or large SUVs.
        - If user gives purpose (e.g. business, vacation), consider suitable categories:
            • Business → Premium, Executive
            • Vacation → SUV, Comfort, Family, Minivan
        - Match as best as possible. Prioritize more relevant cars.
        - If nothing matches exactly, still choose close options, but NEVER suggest cars with fewer seats than needed.
        - Return only a JSON array of IDs like: [12, 33, 55]

        NEVER include text, explanation, formatting. Raw JSON only.
        `.trim(),
  },
  {
    role: "user",
    content: `Available cars:\n${carList}\n\nUser request: ${userInput}`,
  },
];

const completion = await openai.chat.completions.create({
  model: "gpt-4o",
  messages,
  temperature: 0.3,
});

const textRaw = completion.choices[0].message.content || "[]";
console.log("🧠 GPT response:", textRaw);

const match = textRaw.match(/\[(\s*\d+\s*,?)*\]/);
const extracted = match?.[0] ?? "[]";
const ids = safeParseJsonArray(extracted);

if (!ids.length) {
  console.warn("⚠️ No IDs parsed, using fallback IDs.");
```

Рисунок 3.2.11 – Формування prompt-запиту для моделі GPT-4o

Рисунок 3.2.12 демонструє frontend-реалізацію компонента SearchAI, де формується запит на основі даних, заповнених у формі (бюджет, ціль, тип пального тощо). Запит відправляється на бекенд через fetch до /api/search-ai, після чого очікується відповідь із підібраними автомобілями.

У разі помилки або відсутності результатів система повідомляє користувача про невдалий підбір.

Цей підхід дозволив реалізувати семантичний пошук авто, де замість класичної фільтрації за вибраними опціями, користувач може просто описати свої потреби у довільній формі, а система самостійно інтерпретує запит і поверне найвідповідніші варіанти.

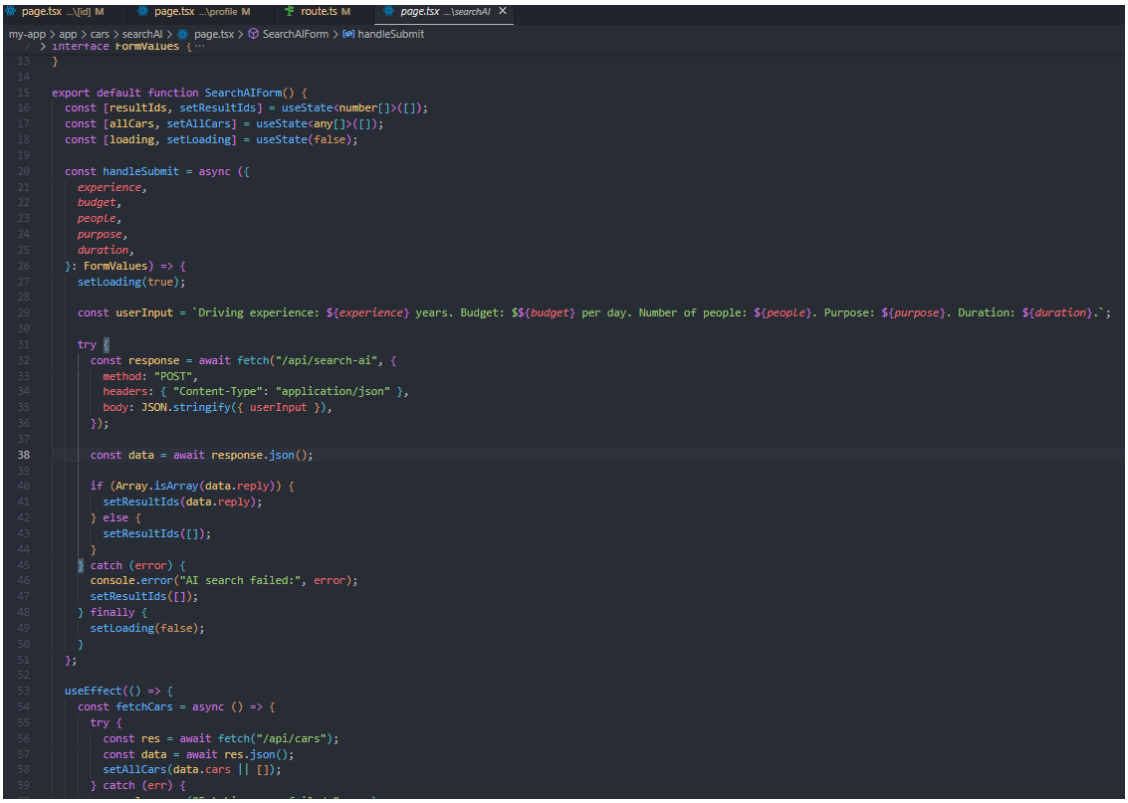


Рисунок 3.2.12 – Frontend-реалізація компонента SearchAI

На рисунку 3.2.13 представлено фінальну частину логіки сторінки пошуку за допомогою штучного інтелекту (SearchAI.tsx). Цей компонент відповідає за обробку результатів, отриманих від GPT-4o, та відображення відповідних автомобілів з бази даних.

У блоці useEffect() після рендеру сторінки здійснюється запит до API /api/cars, який повертає список усіх авто для подальшого фільтрування. Збережені автомобілі зберігаються у стані allCars.

У центральній частині компонента реалізовано фільтрацію: filteredCars визначаються шляхом зіставлення car.id з ідентифікаторами, які повернула GPT-модель.

Логіка виводу враховує різні стани: очікування відповіді (loading), наявність результатів, або їхню відсутність.

```
export default function SearchAIForm() {
  const handleSubmit = async ({
        setResultIds(data.reply);
      } else {
        setResultIds([]);
      }
    } catch (error) {
      console.error("AI search failed:", error);
      setResultIds([]);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    const fetchCars = async () => {
      try {
        const res = await fetch("/api/cars");
        const data = await res.json();
        setAllCars(data.cars || []);
      } catch (err) {
        console.error("Fetching cars failed:", err);
      }
    };
    fetchCars();
  }, []);

  const filteredCars = allCars.filter((car) => resultIds.includes(car.id));

  return (
    <div className="max-w-5xl mx-auto mt-10 p-4 border rounded-xl shadow-md">
      <h2 className="text-2xl sm:text-3xl font-semibold mb-6 text-center">
        Find your car with AI
      </h2>

      <AIFormInputs onSubmit={handleSubmit} isLoading={loading} />

      {loading && (
        <p className="text-center text-gray-500 mt-6 animate-pulse">
          Generating AI recommendations...
        </p>
      )}

      {!loading && filteredCars.length > 0 && (
        <div className="mt-10 grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6">
          {filteredCars.map((car) => (
            <CarCard key={car.id} car={car} />
          ))}
        </div>
      )}

      {!loading && resultIds.length > 0 && filteredCars.length === 0 && (
        <p className="text-center text-red-500 mt-6">
          No matching cars found based on your preferences.
        </p>
      )}
    </div>
  );
}
```

Рисунок 3.2.13 – Реалізація компонента сторінки пошуку за допомогою ШІ

3.3 Реалізація інтерфейсу користувача

Для розробки інтерфейсу у даному проєкті використовується бібліотека React у зв’язці з фреймворком Next.js та стилізатором Tailwind CSS. Компонентний підхід дозволив реалізувати адаптивний, сучасний та легко масштабований інтерфейс з повторно використовуваними блоками. Дизайн орієнтований на зручність користувача, логічну структуру та швидкий доступ до основних функцій системи: перегляду автопарку, фільтрації, перегляду деталей авто, взаємодії з рекомендаційним модулем та редагування профілю.

Усі сторінки оптимізовано під різні розміри екранів як для десктопної, так і мобільної версії. Tailwind CSS дозволив реалізувати гнучкий дизайн, теми, анімації (за допомогою Framer Motion) та забезпечити узгоджене візуальне оформлення системи.

На рисунках 3.3.1-3.3.10 представлено вікна інтерфейсу веб-застосунку.

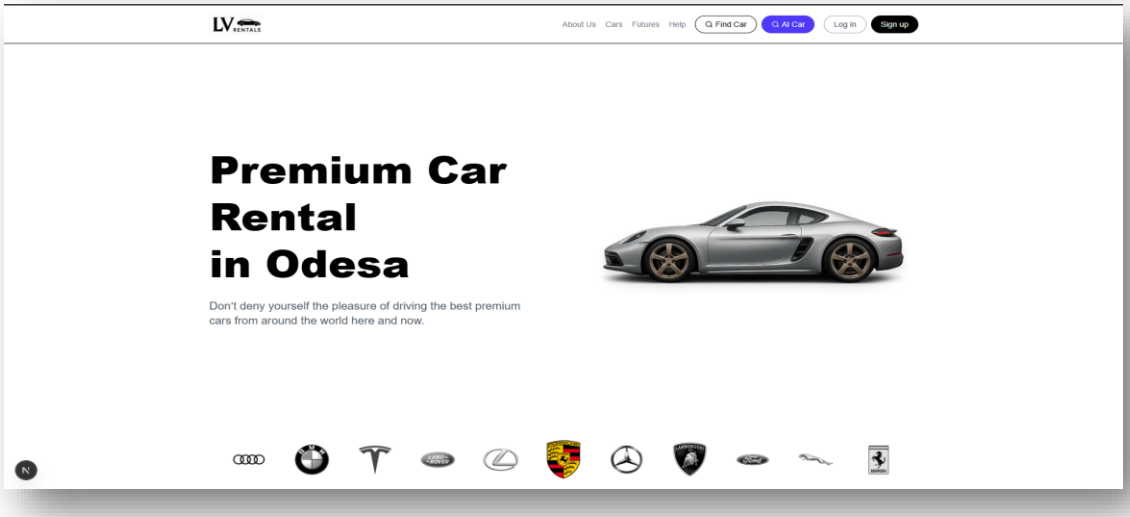


Рисунок 3.3.1 – Головна сторінка веб-застосунку

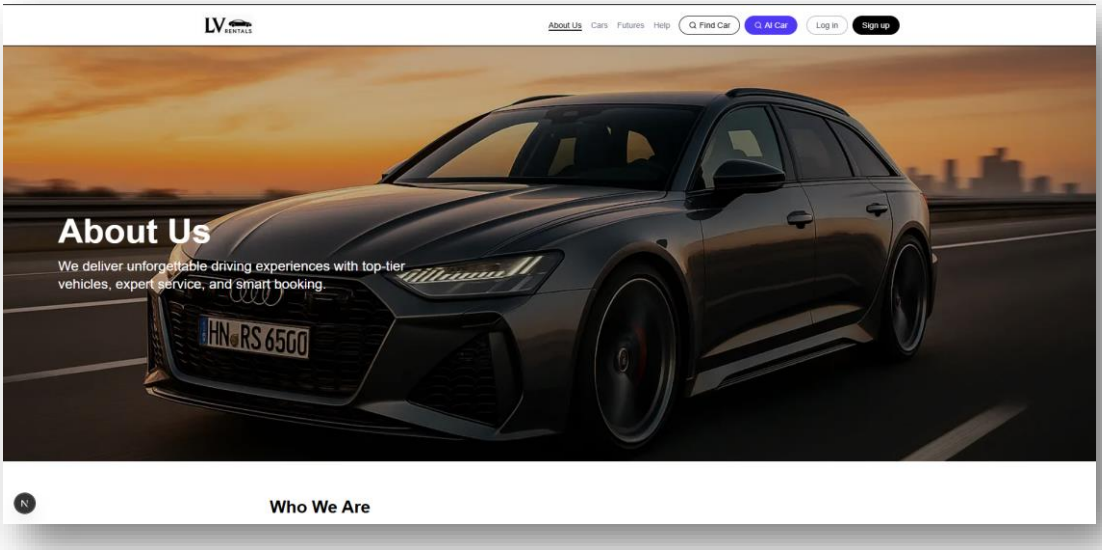


Рисунок 3.3.2 – Сторінка about

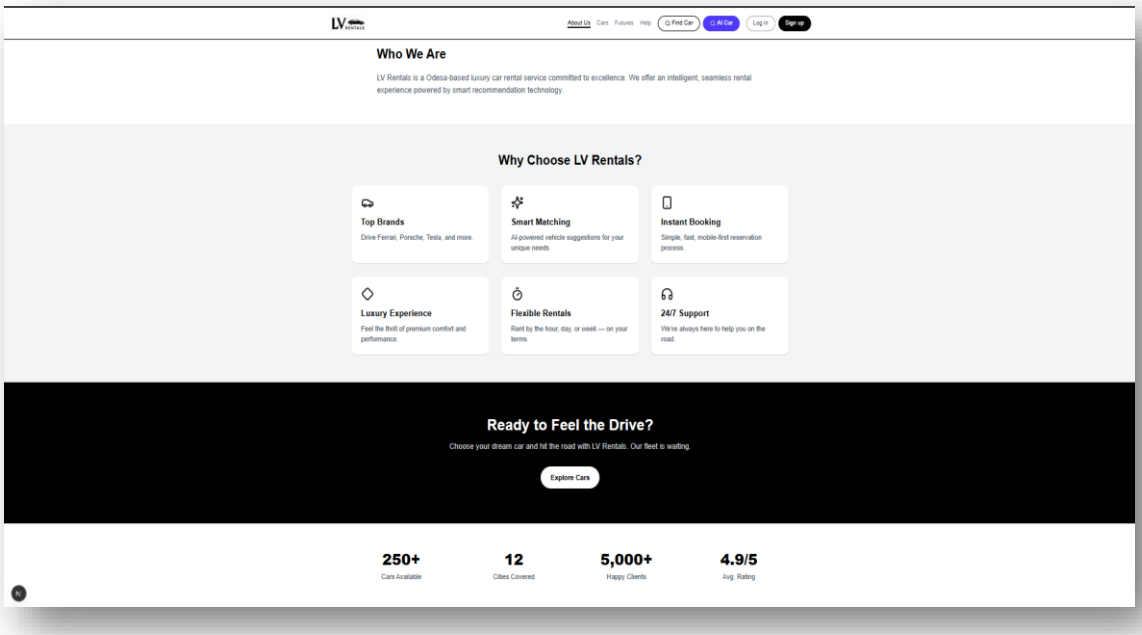


Рисунок 3.3.2.1 – Сторінка about

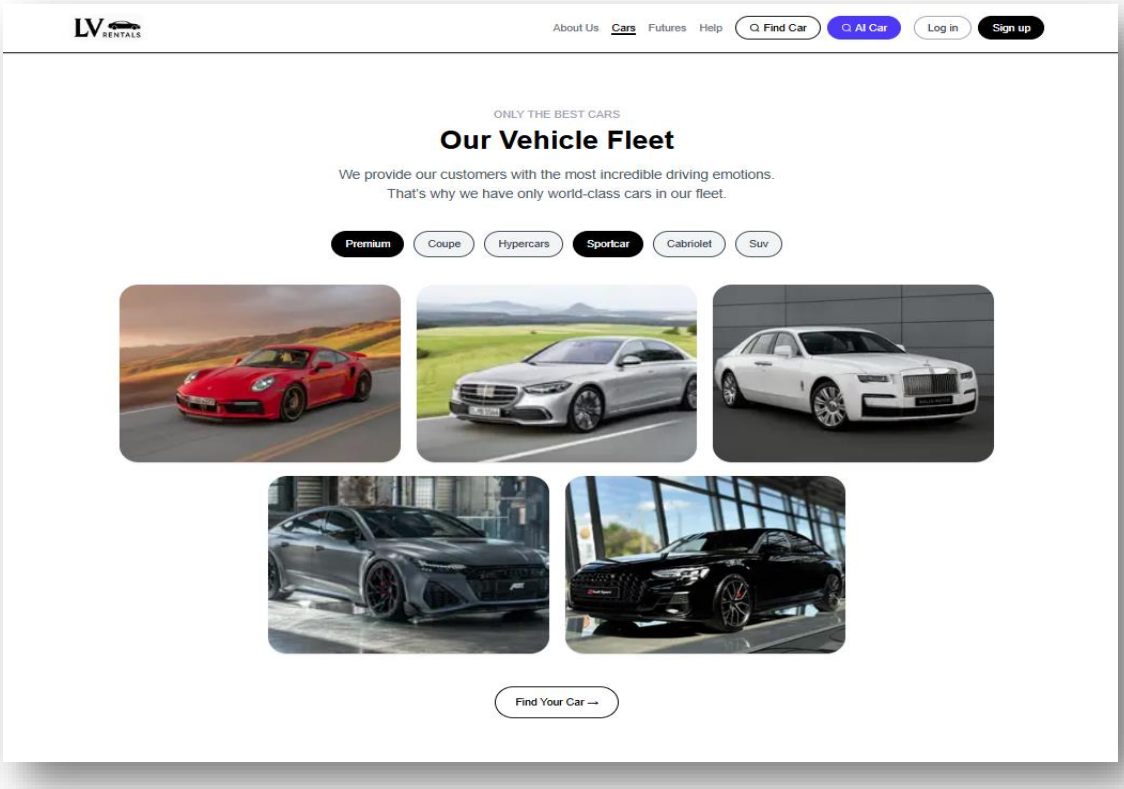


Рисунок 3.3.3 – Галерея авто

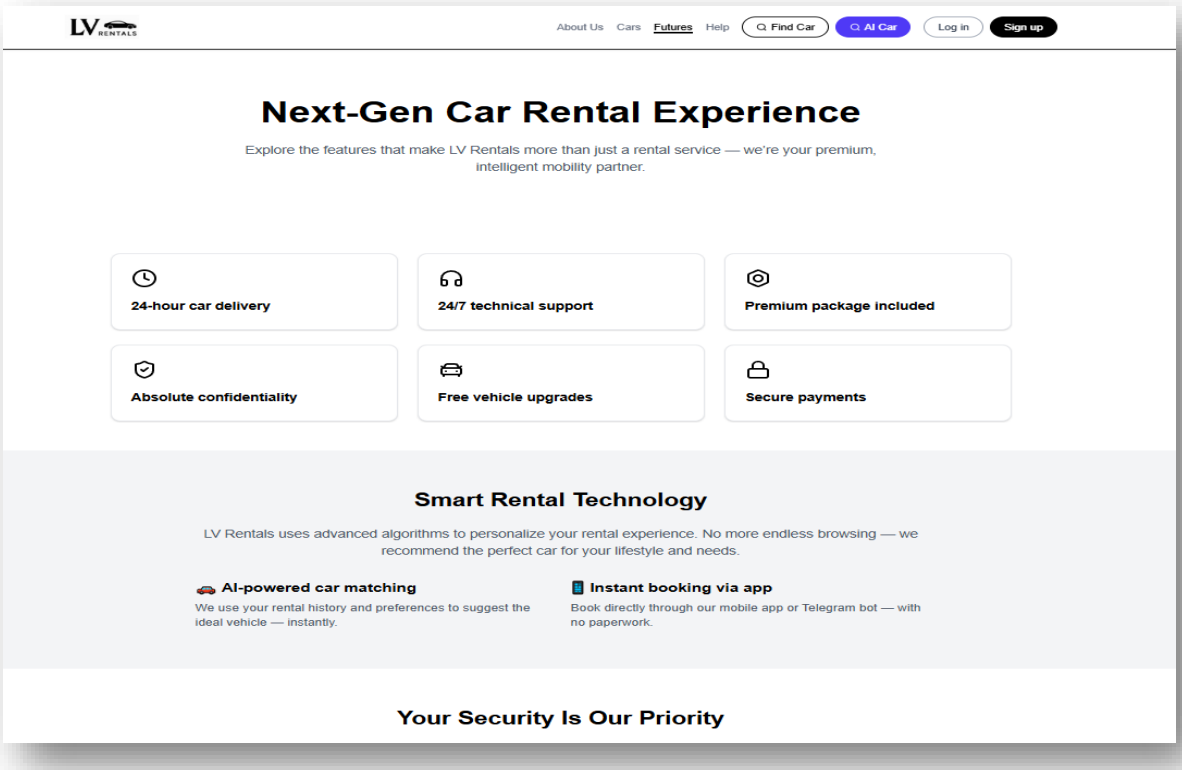


Рисунок 3.3.4 – Сторінка Futures

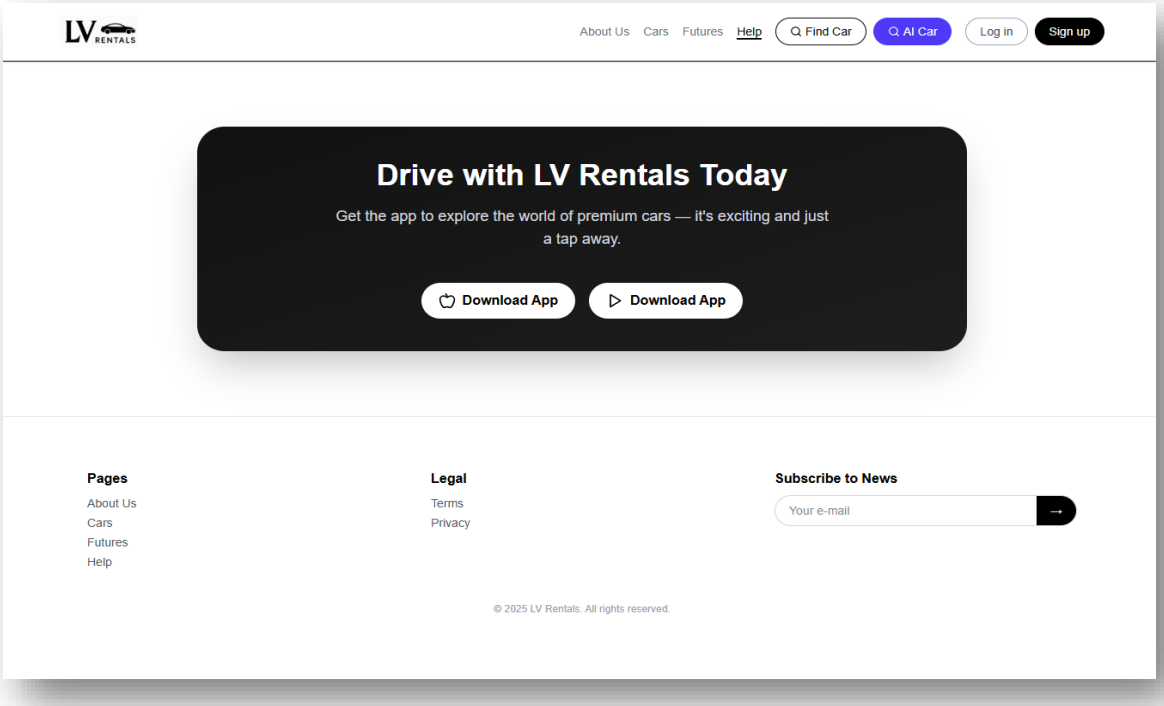


Рисунок 3.3.5 – Сторінка Help

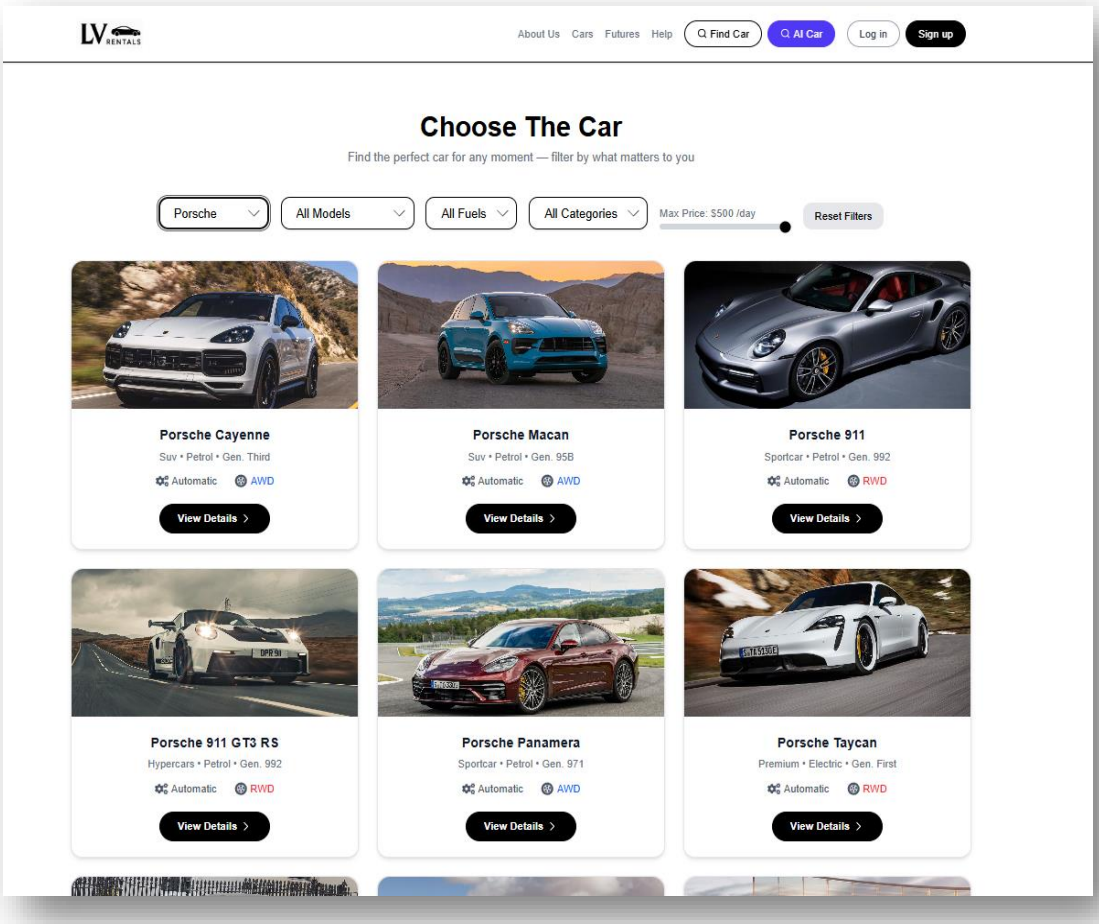


Рисунок 3.3.6 – Сторінка Find Car

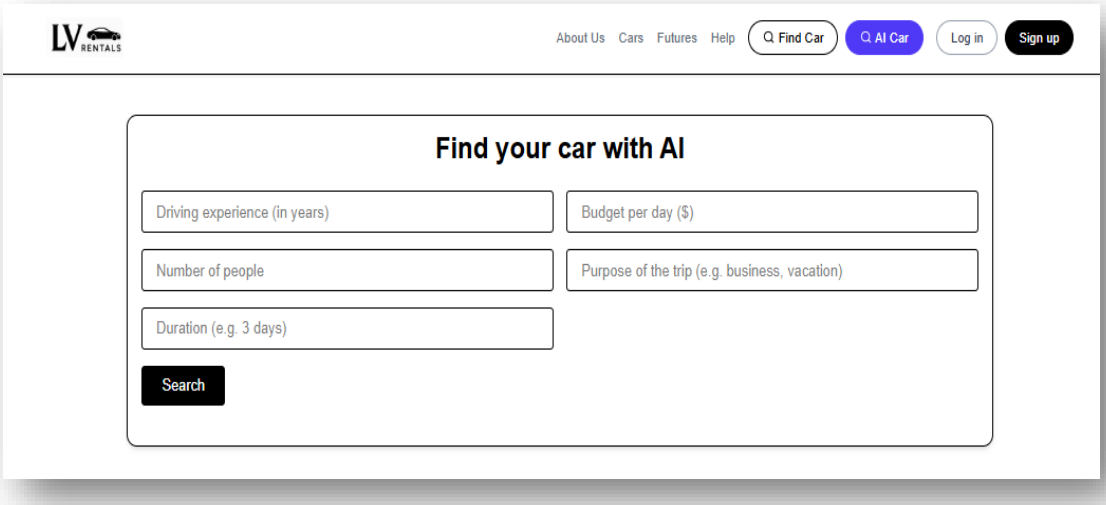


Рисунок 3.3.7 – Сторінка AI Car (до заповнення даних)

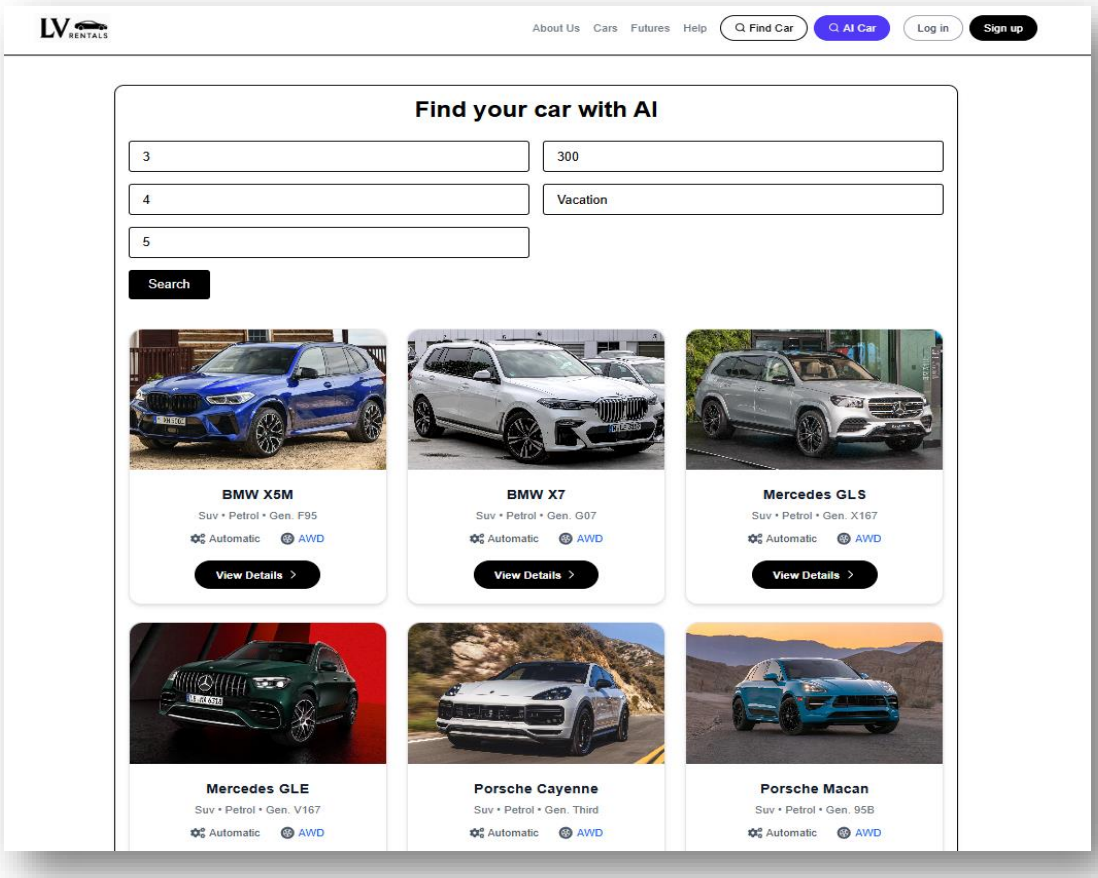


Рисунок 3.3.8 – Сторінка AI Car (після отримання даних)

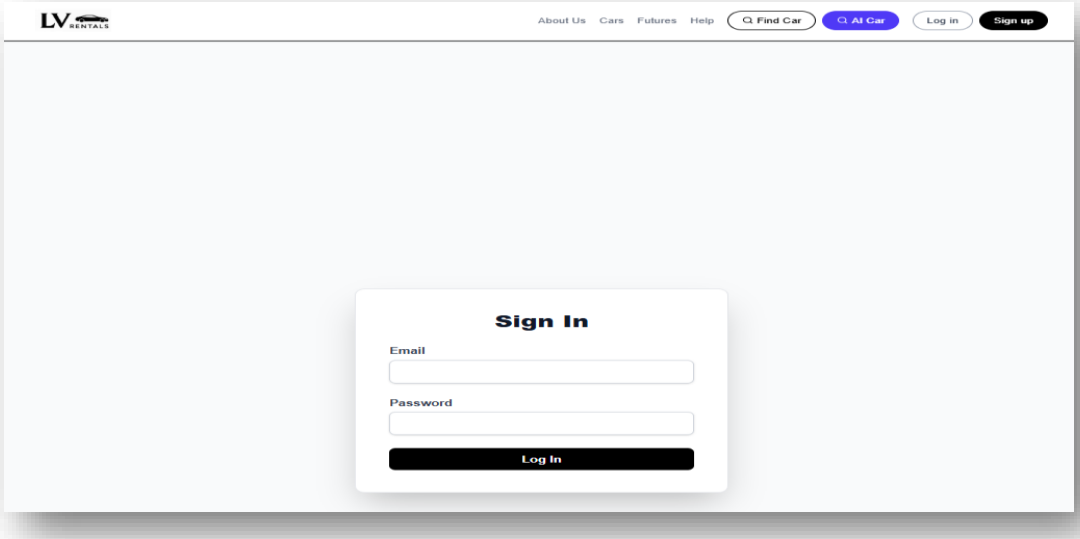


Рисунок 3.3.9 – Log in

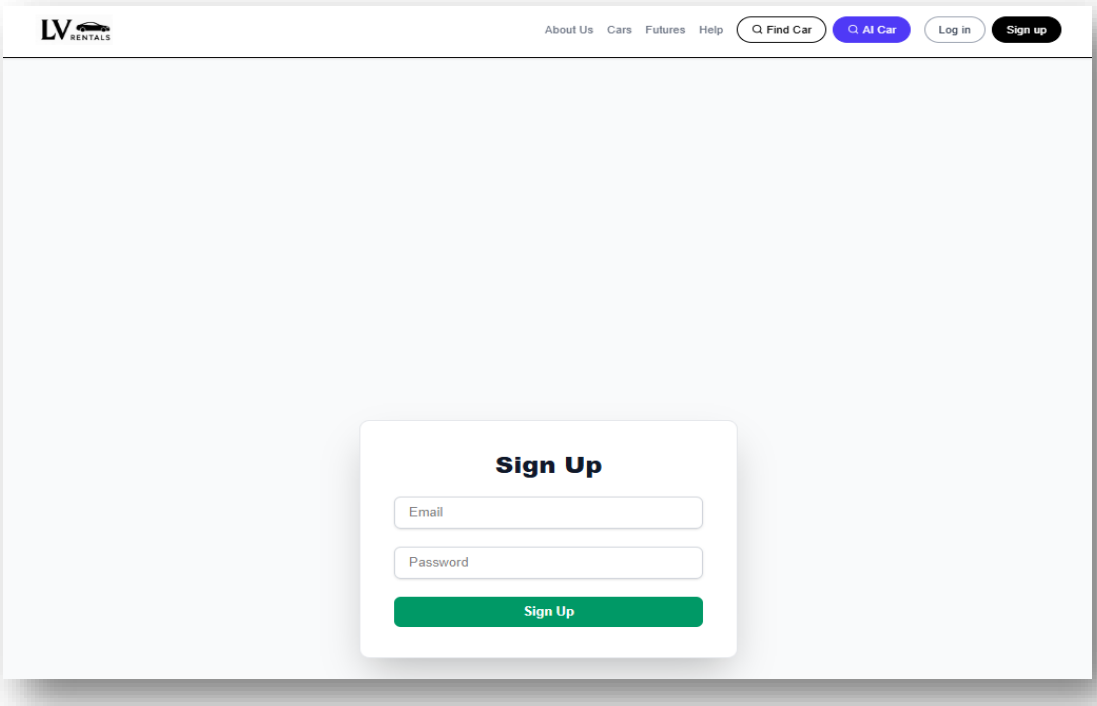


Рисунок 3.3.9 – Sign Up

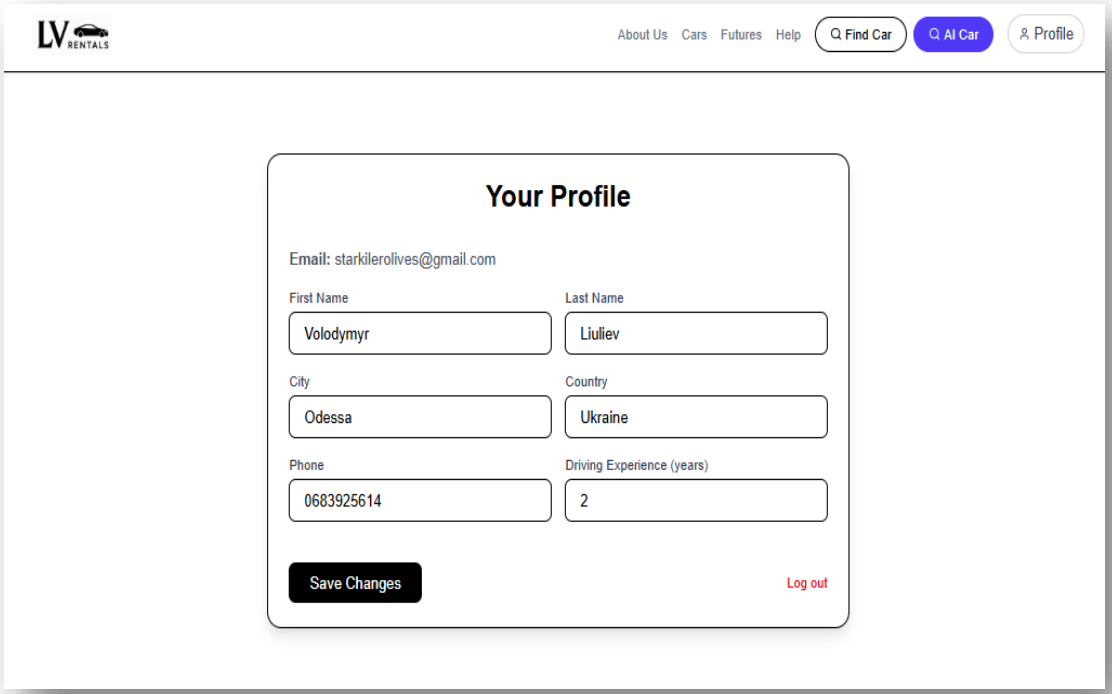


Рисунок 3.3.10 – Сторінка Profile

3.4 Тестування веб-застосунку

Тестування є невід'ємною частиною процесу розробки програмного забезпечення, оскільки воно дозволяє виявляти дефекти, перевіряти коректність функціоналу, забезпечити якість інтерфейсу та гарантувати стабільну роботу застосунку в реальних умовах. У межах розробки веб-застосунку для прокату автомобілів було проаналізовано типові сценарії та реалізовано умовне тестування на рівні логіки та інтерфейсу.

Модульне тестування (Unit Testing) зосереджене на перевірці окремих функцій або методів логіки компонента. У проєкті є кілька прикладів, де тестування логіки можливе без запуску всього інтерфейсу:

1) обробка фільтрів авто (функція filterCars);
2) виведення рекомендацій із AI;
3) оновлення форми профілю користувача.

Протестовано функцію filterCars, яка відповідає за відбір автомобілів за заданими параметрами, такими як бренд і тип палива. Функція не залежить від інтерфейсу, тому є ідеальним кандидатом для модульного тестування. На рисунку 3.4.1 представлено код функції.

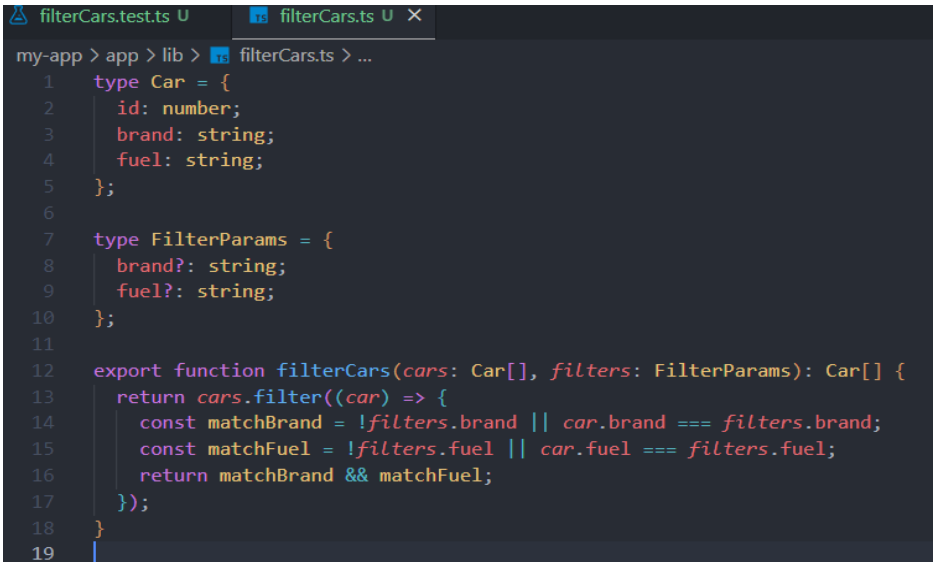


Рисунок 3.4.1 – Код функції filterCars.ts

Для перевірки правильності роботи функції filterCars створено окремий unit-test, який перевіряє, чи повертається лише автомобіль з брендом BMW, якщо задано відповідний фільтр (рис. 3.4.2). Результат виконання тесту представлено на рисунку 3.4.3.

Рисунок 3.4.2 – Unit-test функції filterCars

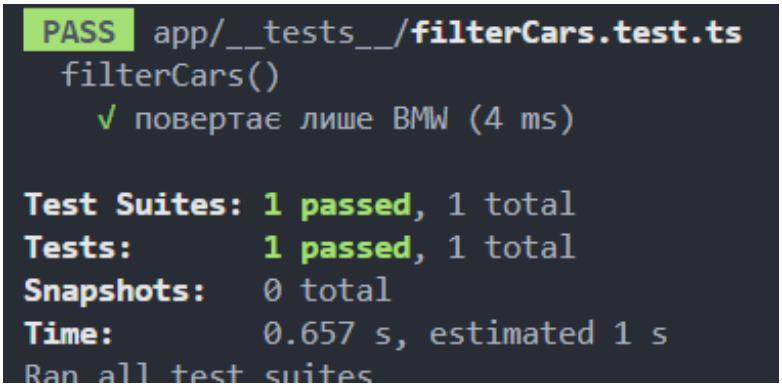


Рисунок 3.4.3 – Результат виконання модульного тесту

Проведено модульне тестування функції getRecommendedCarIds, яка відповідає за обробку результатів, отриманих від AI, та формування списку рекомендованих автомобілів (рис. 3.4.4, 3.4.5). Оскільки ця функція взаємодіє з зовнішнім сервісом GPT (через OpenAI API), для тестування використано

заглушку (fakeGptRespond), яка імітує відповідь мовної моделі на введене користувачем запитання.

Функція getRecommendedCarIds() приймає три аргументи:

1) масив автомобілів із мінімальними даними (id, brand, model);
2) введений користувачем текст (наприклад, опис його потреб);
3) функцію, яка повертає відповідь GPT.

В тесті перевіряється, що після симульованої відповіді GPT (наприклад, "IDs: 1, 3"), функція правильно повертає масив [1, 3], що відповідає рекомендованим автомобілям.

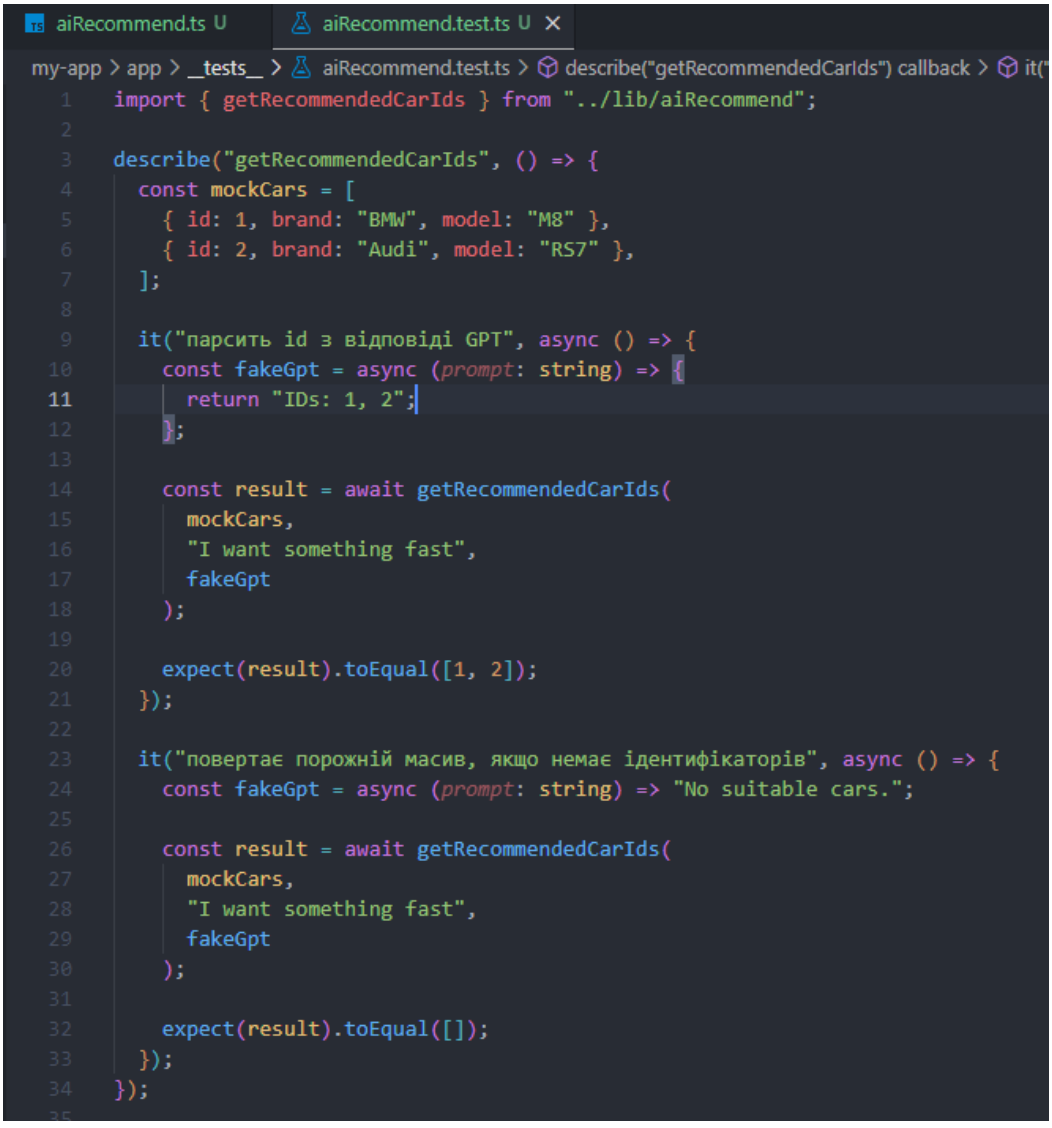


Рисунок 3.4.4 – Фрагмент коду модульного тесту

```
PASS app/__tests__/aiRecommend.test.ts
  getRecommendedCarIds
    √ парсить id з відповіді GPT (6 ms)
    √ повертає порожній масив, якщо немає ідентифікаторів

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.748 s
Ran all test suites matching /aiRecommend.test.ts/i.
```

Рисунок 3.4.5 – Результат виконання тесту

Проведено модульне тестування функції updateUserProfile (рис. 3.4.6, 3.4.7), яка відповідає за оновлення даних профілю користувача у базі даних. Щоб протестувати її поведінку окремо від реальної бази даних, використано фіктивну функцію збереження (fakeSaveFn), яка емулює успішне оновлення.

Основні цілі тестування:

1) перевірка коректності передачі нових даних користувача до збереження;
2) перевірка обробки помилкових ситуацій.

updateUserProfile.test.ts U

my-app > app > __tests__ > updateUserProfile.test.ts > describe("updateUserProfile()") callback

```ts
import { updateUserProfile } from "../lib/profile";

const fakeSaveFn = async (id: string, data: any) => ({
  success: true,
  id,
  data,
});

describe("updateUserProfile()", () => {
  it("успішно оновлює дані користувача", async () => {
    const userId = "user_123";
    const newData = {
      name: "Volodymyr",
      country: "Ukraine",
      drivingExperience: "5 років",
    };

    const result = await updateUserProfile(userId, newData, fakeSaveFn);

    expect(result.success).toBe(true);
    expect(result.data.name).toBe("Volodymyr");
  });

  it("викидає помилку, якщо не передано userId", async () => {
    await expect(updateUserProfile("", {}, fakeSaveFn)).rejects.toThrow(
      "Invalid input"
    );
  });
});
```

Рисунок 3.4.6 – Код тестування UpdateUserProfile

```
PASS app/__tests__/updateUserProfile.test.ts
  updateUserProfile()
    √ успішно оновлює дані користувача (4 ms)
    √ викидає помилку, якщо не передано userId (9 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.638 s
Ran all test suites matching /updateUserProfile/i.
```

Рисунок 3.4.7 – Результат виконання тесту в консолі

Функціональне тестування (Functional Testing) перевіряє виконання веб-застосунком своїх ключових функцій згідно із вимогами. У межах цього тестування проаналізовано декілька сценаріїв взаємодії користувача з інтерфейсом.

Протестовано виведення списку автомобілів за фільтрами. Після застосування фільтрів (марка, модель, категорія, тип пального, діапазон ціни) на сторінці пошуку мають відображатися лише ті авто, які відповідають критеріям користувача.

На рисунку 3.4.8 представлено тестування за вимогами:

1. Бренд Ferrari
2. Вартість менше 700.

Рис. 3.4.8 – Застосування фільтрацій до списку авто

На рисунку 3.4.9 представлено результат тестування – на сторінці вивелося лише відповідне авто. Функція працює правильно.

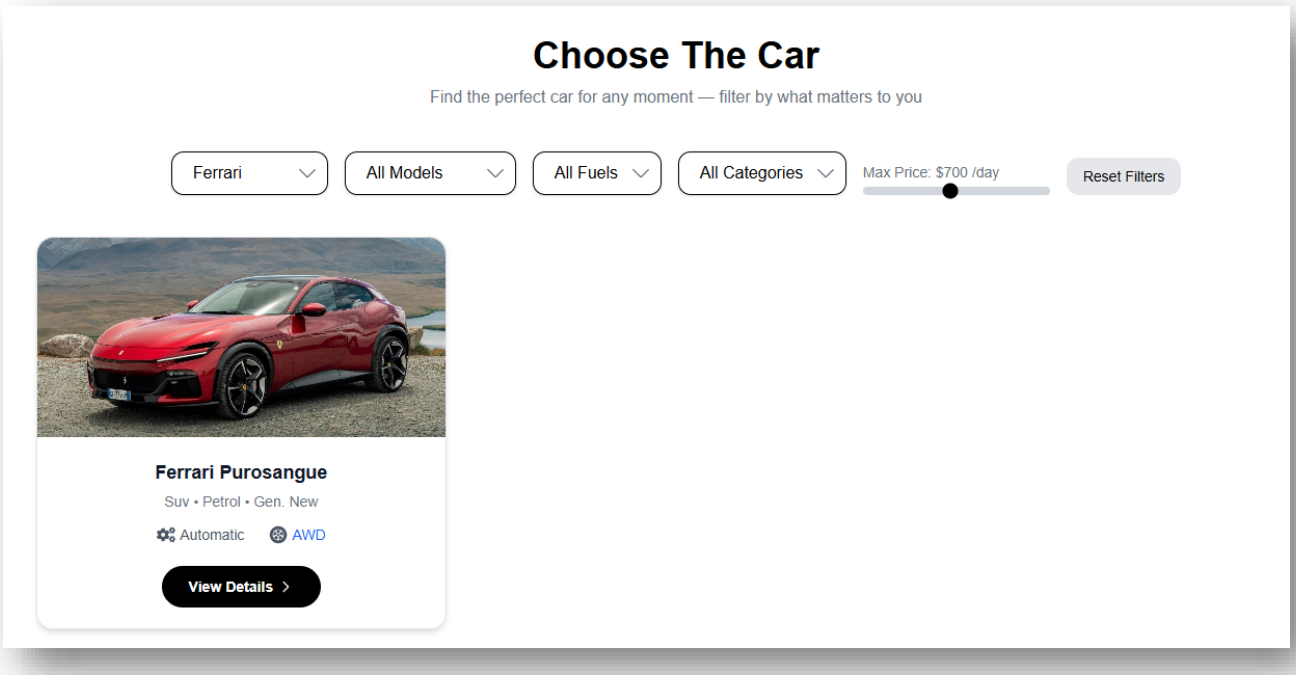


Рис. 3.4.9 – Результат фільтрації – один відповідний автомобіль

Протестовано запит рекомендацій з використанням AI. Після заповнення анкети користувача на основі його вподобань, формується запит до моделі штучного інтелекту, яка повертає список рекомендованих авто.

На рисунку 3.4.10 представлено тестування запиту: заповнено дані про бюджет, тип пального та стиль керування.

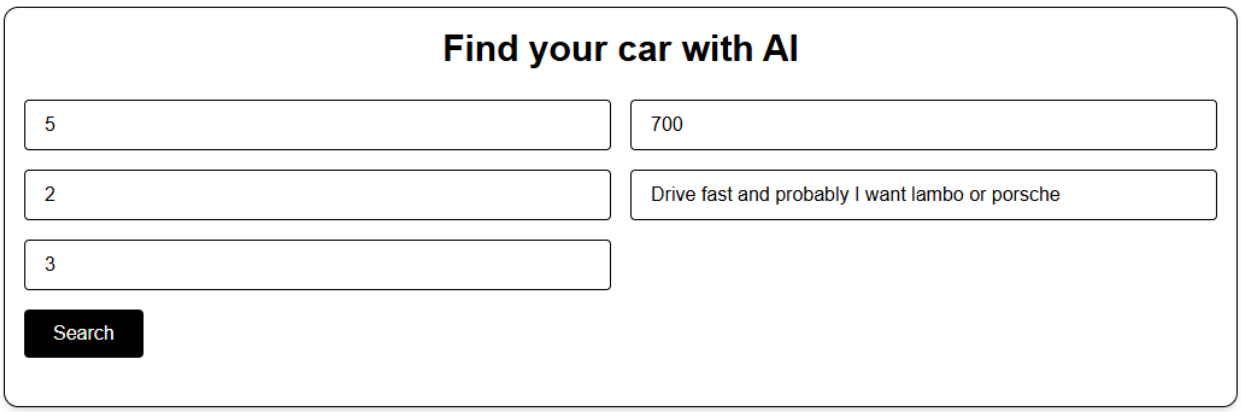


Рис. 3.4.10 – Заповнення анкети користувачем

На рисунку 3.4.11 представлено результат тестування – виводиться авто, яке найкраще відповідає очікуванням користувача.

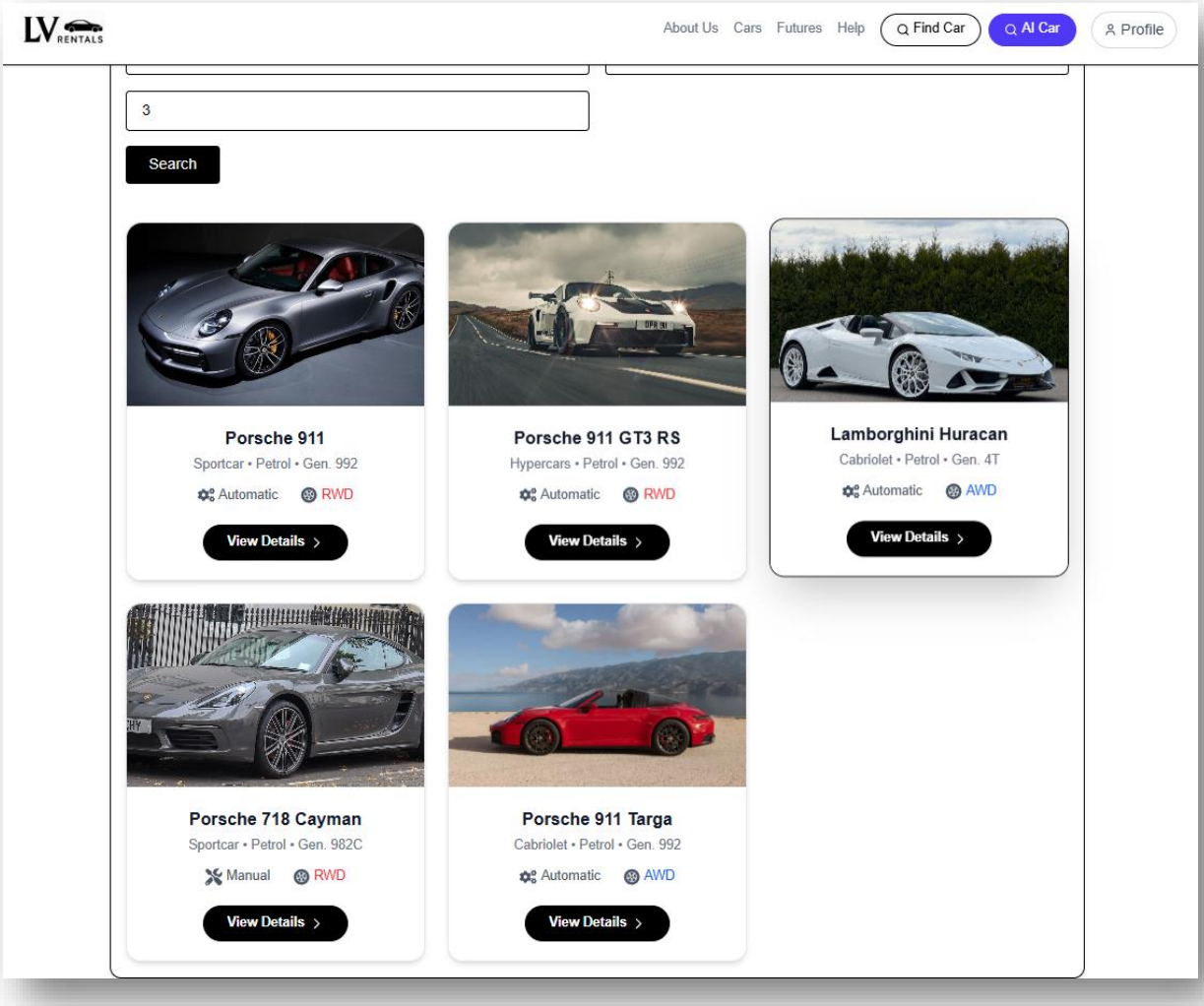


Рис. 3.4.11 – Авто, які рекомендує наша модель штучного інтелекту

Проведено перевірку відображення деталей авто (рис. 3.4.12, 3.4.13). На сторінці конкретного автомобіля перевірено правильність відображення:

1) вартість за день, тиждень і місяць;
2) технічних характеристик (потужність, витрата пального тощо).

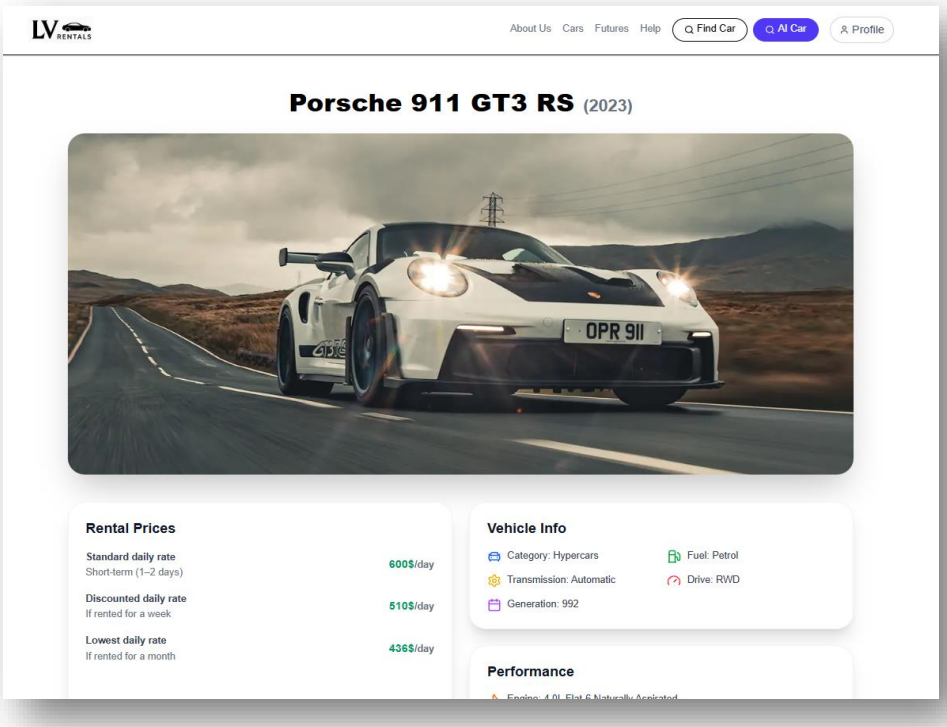


Рис. 3.4.12 – Інтерфейс сторінки з описом авто

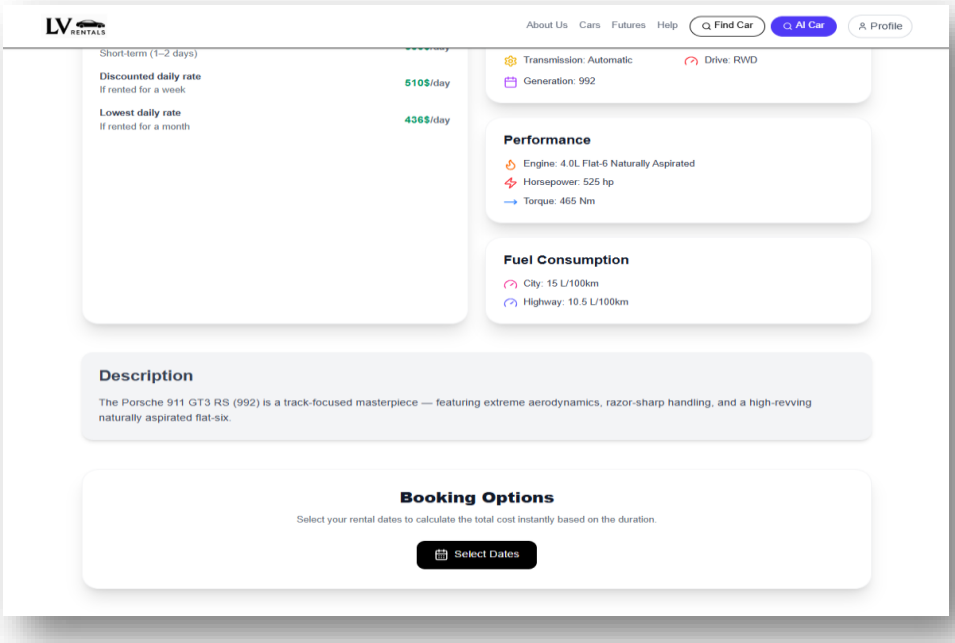


Рис. 3.4.13 – Інтерфейс сторінки з описом авто 2 (елементи підтягуються автоматично)

Джерело: розроблено автором

Висновки до розділу 3

У розділі реалізовано програмну частину інтелектуального веб-застосунку для прокату автомобілів, що містить функціональну логіку та зручний інтерфейс користувача.

Для створення веб-застосунку використано сучасні технології розробки: фреймворк Next.js у поєднанні з мовою TypeScript, бібліотеку стилізації Tailwind CSS, а також хмарну платформу Supabase для організації бази даних. Це забезпечило масштабованість, швидкодію і зручність у розгортанні застосунку.

Розроблено функціонал, який охоплює динамічне завантаження даних з бази, фільтрацію автомобілів за ключовими параметрами, відображення розширеної інформації про кожне авто, а також можливість формування рекомендацій на основі введених користувачем даних. Окрему увагу приділено побудові інтуїтивно зрозумілого, адаптивного інтерфейсу, що забезпечує зручну взаємодію користувача із системою.

Проведено модульне тестування логічних функцій, зокрема фільтрації автівок, обробки рекомендацій та оновлення профілю користувача. Виконано функціональне тестування ключових сценаріїв взаємодії з інтерфейсом. Отримані результати засвідчили коректну роботу основних модулів і підтвердили готовність системи до використання.

Загалом, реалізований програмний продукт демонструє відповідність функціональним вимогам, надійність логіки та технічну цілісність архітектури, що дозволяє надалі розширювати систему з урахуванням нових бізнес-завдань і потреб користувачів.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено інтелектуальний веб-застосунок для системи прокату автомобілів, що дозволяє користувачам здійснювати ефективний пошук і підбір транспортного засобу відповідно до індивідуальних потреб, бюджету та уподобань.

Проаналізовано предметну область автопрокату, визначено основні функціональні вимоги до системи, а також проведено огляд аналогічних рішень на ринку. На основі цього обґрунтовано доцільність створення інноваційного сервісу з використанням штучного інтелекту для підвищення точності рекомендацій і покращення користувацького досвіду.

Розроблено архітектуру системи, що містить клієнтську частину, серверну логіку та взаємодію з хмарною базою даних Supabase. Для реалізації інтерфейсу користувача застосовано фреймворк Next.js, мову TypeScript та бібліотеку Tailwind CSS.

Інтелектуальну складову системи реалізовано шляхом інтеграції моделі GPT, яка на основі вхідних даних користувача формує відповідні рекомендації автомобілів. Додатково реалізовано багатофункціональну фільтрацію за параметрами авто: марка, модель, категорія, тип пального, ціна за добу.

Проведено модульне тестування логіки фільтрації, обробки рекомендацій та оновлення профілю користувача. Виконано функціональне тестування взаємодії з основними елементами інтерфейсу.

Результати роботи представлено на 22 Всеукраїнській конференції студентів та молодих науковців «Інформатика, інформаційні системи та технології» (м. Одеса, 25.04.2025 року) [22].

Таким чином, розроблена система демонструє здатність покращувати процес оренди автомобілів завдяки використанню сучасних веб-технологій та елементів штучного інтелекту, а також має потенціал до масштабування, впровадження додаткових сервісів та інтеграції з мобільними платформами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statista. Global Car Rental Market Revenue 2023. URL: https://www.statista.com/statistics/1106013/global-car-rental-market-revenue/ (дата звернення: 03.03.2025)
2. Fleet and revenue management in car rental companies: A literature review and an integrated conceptual framework. *Omega*, Elsevier (2016). URL:https://www.sciencedirect.com/science/article/pii/S0305048316300342 (дата звернення: 07.03.2025)
3. Ricci F., Rokach L., Shapira B. *Recommender Systems Handbook*. Springer, 2015.
4. Rentalcars.com. URL: https://www.rentalcars.com (дата звернення: 10.03.2025)
5. Getaround. URL: https://www.getaround.com (дата звернення: 13.03.2025)
6. BLS – Сервіс оренди в Україні. URL: https://bls.ua/ua (дата звернення: 16.03.2025)
7. scikit-learn documentation – Nearest Neighbors. URL: https://scikit-learn.org/stable/modules/neighbors.html (дата звернення: 20.03.2025)
8. scikit-learn documentation – K-means clustering. URL: https://scikit-learn.org/stable/modules/clustering.html#k-means (дата звернення: 23.03.2025)
9. scikit-learn documentation – Decision Trees. URL: https://scikit-learn.org/stable/modules/tree.html (дата звернення: 27.03.2025)
10. React Documentation. URL: https://react.dev/learn (дата звернення: 01.04.2025)
11. TypeScript Handbook. URL: https://www.typescriptlang.org/docs/ (дата звернення: 04.04.2025)
12. Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 08.04.2025)

13. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (дата звернення: 12.04.2025)
14. Supabase Documentation. URL: https://supabase.com/docs (дата звернення: 17.04.2025)
15. Framer Motion Documentation. URL: https://www.framer.com/motion/ (дата звернення: 21.04.2025)
16. Jest – JavaScript Testing Framework. URL: https://jestjs.io/docs/getting-started (дата звернення: 26.04.2025)
17. OpenAI API Reference. URL: https://platform.openai.com/docs (дата звернення: 03.05.2025)
18. PostgreSQL Official Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 11.05.2025)
19. W3C HTML & CSS Standards. URL: https://www.w3.org/standards/webdesign/ (дата звернення: 17.05.2025)
20. MDN Web Docs (JavaScript, DOM, APIs). URL: https://developer.mozilla.org/ (дата звернення: 22.05.2025)
21. GitHub Docs – Working with Repositories. URL: https://docs.github.com/en/repositories (дата звернення: 28.05.2025)
22. Інформатика, інформаційні системи та технології: тези доповідей двадцять другої всеукраїнської конференції студентів і молодих науковців // Люлєв В.В., Єпік М.О. – Одеса, 25 квітня 2025 р. – Одеса, 2025. – с. 153-155

ДОДАТОК А

Основні фрагменти коду

Код компоненту CarFilter

```
 useEffect(() => {
   const filtered = cars.filter((car) => {
     return (
       (!selectedBrand || car.brand === selectedBrand) &&
       (!selectedModel || car.model === selectedModel) &&
       (!selectedFuel || car.fuel === selectedFuel) &&
       (!selectedCategory || car.category === selectedCategory) &&
       car.price_day >= priceRange[0] &&
       car.price_day <= priceRange[1]
     );
   });
   onFilter(filtered);
 }, [
   selectedBrand,
   selectedModel,
   selectedFuel,
   selectedCategory,
   priceRange,
 ]);
 const resetFilters = () => {
   setSelectedBrand("");
   setSelectedModel("");
   setSelectedFuel("");
   setSelectedCategory("");
   setPriceRange([0, 1500]);
 };
 return (….)
}
```

Код компоненту aiRecommend

```
export async function getRecommendedCarIds(
  cars: { id: number; brand: string; model: string }[],
  userInput: string,
  gptRespondFn: (prompt: string) => Promise<string>
): Promise<number[]> {
  const carList = cars
    .map((car) => `ID: ${car.id}, ${car.brand} ${car.model}`)
    .join("\n");

  const prompt = `Available cars:\n${carList}\n\nUser input: ${userInput}\nReturn only IDs.`;
  const reply = await gptRespondFn(prompt);
  const ids = reply.match(/\d+/g)?.map((id) => parseInt(id, 10)) ?? [];
  return ids;
}
```

ДОДАТОК Б
Приклади інтерфейсу користувача

Код компоненту Header

```
"use client";
import { motion } from "framer-motion";
import LogoBlock from "./LogoBlock";
import NavMenu from "./NavMenu";
import SearchButtons from "./SearchButtons";
import AuthButtons from "./AuthButtons";
export default function Header() {
  return (
    <motion.header
      initial={{ y: -40, opacity: 0 }}
      animate={{ y: 0, opacity: 1 }}
      transition={{ duration: 0.6 }}
      className="border-b bg-white shadow-sm sticky top-0 z-50"
    >
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-4 flex flex-wrap justify-
between items-center gap-4">
        <LogoBlock />
        <nav className="flex flex-wrap gap-4 items-center justify-end flex-1">
          <NavMenu />
          <SearchButtons />
          <AuthButtons />
        </nav>
      </div>
    </motion.header>
  );
}
```

Код компоненту Profile

```
"use client";
import { useEffect, useState } from "react";
import { useRouter } from "next/navigation";
import { supabase } from "@/app/lib/supabaseClient";
export default function ProfilePage() {
  const router = useRouter();
  const [loading, setLoading] = useState(true);
  const [saving, setSaving] = useState(false);
  const [userEmail, setUserEmail] = useState<string | null>(null);

  const [form, setForm] = useState({
    firstName: "",
    lastName: "",
    city: "",
    country: "",
```

```
  phone: "",
  drivingExperience: "",
});

const [message, setMessage] = useState("");

useEffect(() => {
  const getUser = async () => {
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (!user || error || !user.email) {
      router.push("/auth/login");
      return;
    }

    setUserEmail(user.email);

    const { data: profile, error: profileError } = await supabase
      .from("users")
      .select("*")
      .eq("email", user.email)
      .single();

    if (profile) {
      setForm({
        firstName: profile.first_name || "",
        lastName: profile.last_name || "",
        city: profile.city || "",
        country: profile.country || "",
        phone: profile.phone || "",
        drivingExperience: profile.driving_experience || "",
      });
    } else {
      await supabase.from("users").insert([
        {
          email: user.email,
          first_name: "",
          last_name: "",
          city: "",
          country: "",
          phone: "",
          driving_experience: "",
        },
      ]);
    }
```

Код компоненту Profile. Аркуш 2

```
    setLoading(false);
  };

  getUser();
}, [router]);

const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const { name, value } = e.target;
  setForm((prev) => ({ ...prev, [name]: value }));
};

const handleSave = async () => {
  if (!userEmail) return;
  setSaving(true);
  const { error } = await supabase.from("users").upsert(
    {
      email: userEmail,
      first_name: form.firstName,
      last_name: form.lastName,
      city: form.city,
      country: form.country,
      phone: form.phone,
      driving_experience: form.drivingExperience,
    },
    { onConflict: "email" }
  );

  setMessage(
    error ? "❌ Update failed." : "✅ Profile updated successfully."
  );
  setSaving(false);
  setTimeout(() => setMessage(""), 3000);
};

const handleLogout = async () => {
  await supabase.auth.signOut();
  router.push("/auth/login");
};

if (loading)
  return (
    <p className="text-center mt-10 text-gray-500">Loading profile...</p>
  );
return (
  <div className="max-w-2xl mx-auto mt-20 p-6 border rounded-2xl shadow-lg bg-white">
    <h1 className="text-3xl font-bold text-center mb-8">Your Profile</h1>
```

Код компоненту Profile. Аркуш 3

```
      <div className="space-y-4">
        <div className="text-gray-600">
          <strong>Email:</strong> {userEmail}
        </div>

        <div className="grid grid-cols-1 sm:grid-cols-2 gap-4">
          {[
            { label: "First Name", name: "firstName" },
            { label: "Last Name", name: "lastName" },
            { label: "City", name: "city" },
            { label: "Country", name: "country" },
            { label: "Phone", name: "phone" },
            { label: "Driving Experience (years)", name: "drivingExperience" },
          ].map((field) => (
            <div key={field.name}>
              <label className="block text-sm font-medium mb-1 text-gray-700">
                {field.label}
              </label>
              <input
                name={field.name} value={(form as any)[field.name]} onChange={handleChange}
                placeholder={`Enter ${field.label.toLowerCase()}`}
                className="w-full px-4 py-2 border rounded-lg focus:outline-none focus:ring-2
focus:ring-black/60"
              />
            </div>
          ))}
        </div>
        <div className="flex flex-col sm:flex-row justify-between items-center gap-4 pt-6">
          <button
            onClick={handleSave}
            disabled={saving}
            className="bg-black text-white px-6 py-2 rounded-lg hover:bg-gray-800
disabled:opacity-50"
          >
            {saving ? "Saving..." : "Save Changes"}
          </button>

          <button
            onClick={handleLogout}
            className="text-red-600 hover:underline text-sm"
          >
            Log out
          </button>
        </div>

        {message && (
          <div className="mt-4 text-center text-sm text-blue-600 animate-pulse">
            {message}</div>)}</div></div>);}
```

Код компоненту Profile. Аркуш 4