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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

«Сайт оголошень з продажу автомобілів»

Виконав студент групи ІПЗ-21-5
ГЕЙНА Владислав Сергійович

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
ВЛАСЕНКО Олег Васильович

Житомир – 2025

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

«ЗАТВЕРДЖУЮ»
Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«23» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ГЕЙНА Владислав Сергійович**
Керівник роботи: **ЧИЖМОТРЯ Олексій Володимирович**
Тема роботи: **«Сайт оголошень з продажу автомобілів»**,
затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**
Вихідні дані для роботи: об’єктом дослідження є веб-платформа для розміщення оголошень про продаж автомобілів. Предметом дослідження є підходи та засоби розробки сучасного веб-застосунку, що забезпечує зручне додавання, перегляд, пошук оголошень, а також взаємодію між користувачами.
Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	**Консультант**	Завдання видав	Завдання прийняв
1	Чижмотря О. В.	14.02.2025	14.02.2025
2	Чижмотря О. В.	14.02.2025	14.02.2025
3	Чижмотря О. В.	14.02.2025	14.02.2025

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з веб-додатку для розміщення оголошень про продажу автомобілів та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить текстову частину, викладену на 84 сторінках друкованого тексту. Список використаних джерел містить 39 найменувань і займає 3 сторінки. У роботі наведено 53 ілюстрації та 15 таблиць. Також у роботі є 10 сторінок додатків. Загальний обсяг роботи – 94 сторінки.

Головною метою є створення сучасного веб-застосунку для розміщення оголошень про продаж автомобілів, який забезпечує зручне керування оголошеннями користувачами, швидкий та ефективний пошук за різними критеріями, інтерактивну взаємодію між продавцями та покупцями через чат, а також надає можливості модерації контенту для підтримки якості та безпеки платформи.

У ході роботи було проведено аналіз напрямів використання інформаційних технологій, сформульовано постановку задачі та виконано огляд існуючих програмних рішень для сервісів розміщення оголошень. У результаті обрано архітектуру системи, що відповідає вимогам проєкту. Визначено варіанти використання системи та побудовано її об’єктно-орієнтовану структуру. Здійснено розробку бази даних і реалізацію алгоритмів роботи, включаючи функціонал інтерфейсу та налаштування параметрів веб-застосунку. Усі етапи проєкту пройшли детальне тестування для забезпечення стабільності та надійності платформи, що підтверджено результатами, наведеними в третьому розділі.

КЛЮЧОВІ СЛОВА: REACT.JS, NODE.JS, EXPRESS, TYPESCRIPT, POSTGRESQL, REDIS, WEBSOCKET, JWT, GOOGLE АВТОРИЗАЦІЯ, AWS S3, REST API, ОГОЛОШЕННЯ, МОДЕРАЦІЯ, VIN-КОД.

ABSTRACT

The bachelor's qualification thesis consists of a web application for posting car sale advertisements and an explanatory note. The explanatory note comprises 84 pages of printed text. The list of references includes 39 sources and occupies 3 pages. The work contains 53 illustrations and 15 tables. Additionally, there are 10 pages of appendices. The total volume of the thesis is 94 pages.

The main goal of this work is to create a user-friendly platform that allows efficient management of advertisements, fast and advanced search by various criteria, interactive communication between sellers and buyers via chat, as well as moderation tools to ensure content quality and platform security.

The work includes a comprehensive analysis of current information technologies, formulation of the problem statement, and a review of existing software solutions for classified advertisement services. Based on this analysis, an appropriate system architecture was selected to meet the project requirements. Use cases were defined, and an object-oriented system structure was designed. The development of the database and implementation of system algorithms, including interface functionality and configuration settings, were carried out. All stages of the project underwent thorough testing to ensure platform stability and reliability, confirmed by the results presented in Chapter 3.

KEYWORDS: REACT.JS, NODE.JS, EXPRESS, TYPESCRIPT, POSTGRESQL, REDIS, WEBSOCKET, JWT, GOOGLE AUTHORIZATION, AWS S3, REST API, AD, MODERATION, VIN-CODE.

ЗМІСТ

РЕФЕРАТ 3
ABSTRACT 4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 6
ВСТУП 7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ 9
1.1. Постановка задачі 9
1.2. Аналіз аналогів програмного продукту 11
1.3. Вибір архітектури веб-орієнтованої системи 16
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення 19
Висновки до першого розділу 21
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ 23
2.1. Визначення варіантів використання та об’єктно-орієнтованої структури системи 23
2.2. Розробка бази даних системи 31
2.3. Проектування та реалізація алгоритмів роботи системи 37
2.4. Реалізація функціоналу системи 40
Висновки до другого розділу 51
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ІЗ СЕРВІСОМ 52
3.1. Порядок встановлення та налаштування параметрів системи 52
3.2. Структура інтерфейсу веб-застосунку 54
3.3. Тестування роботи програмного забезпечення 74
Висновки до третього розділу 78
ВИСНОВКИ 80
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 82
ДОДАТКИ 85

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

СКБД – система керування базами даних

SQL – Structured Query Language (структурована мова запитів)

API – Application Programming Interface (прикладний програмний інтерфейс)

REST – Representational State Transfer (передача репрезентативного стану)

URL – Uniform Resource Locator (уніфікований локатор ресурсу)

MVVM – Model-View-ViewModel (модель–вигляд–модель представлення)

ORM – Object-Relational Mapper (об'єктно-реляційний мапер)

XSS – Cross-Site Scripting (міжсайтове скриптування)

HTTP – HyperText Transfer Protocol (протокол передавання гіпертексту)

DOM – Document Object Model (об’єктна модель документа)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

AWS – Amazon Web Services (веб-сервіси Amazon)

DTO – Data Transfer Object (об'єкт передачі даних)

JWT – JSON Web Token (веб-токен у форматі JSON)

UI/UX – User Interface / User Experience (інтерфейс користувача / досвід користувача)

VIN – Vehicle Identification Number (унікальний ідентифікаційний номер транспортного засобу)

ВСТУП

Актуальність теми даної роботи зумовлена постійним зростанням кількості автомобілів та попиту на їх купівлю-продаж. В сучасному світі автомобіль є не лише засобом пересування, а й важливою складовою повсякденного життя, що безпосередньо впливає на розвиток транспортної інфраструктури та економіки. Водночас процес купівлі-продажу транспортних засобів залишається складним через велику кількість офлайн-оголошень, потребу перевірки достовірності інформації та складність швидкого пошуку релевантних пропозицій.

Розвиток інтернет-технологій докорінно змінив підхід до продажу автомобілів, надавши можливість розміщувати оголошення в онлайн-просторі. Спеціалізовані веб-платформи для купівлі-продажу авто стають все більш популярними завдяки своїй зручності, доступності та широкому вибору пропозицій. Вони дозволяють користувачам швидко знаходити необхідний транспортний засіб, порівнювати характеристики і ціни, а також безпосередньо взаємодіяти між собою. Крім того, такі платформи сприяють підвищенню прозорості ринку і зниженню ризиків шахрайства завдяки додатковим механізмам перевірки і модерації контенту.

Метою даної роботи є створення сучасного веб-застосунку для розміщення оголошень про продаж автомобілів, що забезпечує зручне керування оголошеннями, швидкий та ефективний пошук за різними критеріями, інтерактивну взаємодію між користувачами, а також надає можливості контролю якості та безпеки інформації. Особливу увагу приділено також забезпеченню доступності сервісу через інтеграцію з додатковими каналами комунікації, що дозволяє охопити ширшу аудиторію і зробити використання платформи більш комфортним для різних груп користувачів.

Практичними завданнями кваліфікаційної роботи є:

- проведення аналізу теоретичних і практичних вимог до веб-сайтів оголошень про продаж автомобілів;

- визначення структури та архітектури системи, що відповідає сучасним стандартам;
- оцінка існуючих аналогічних веб-рішень і визначення їх сильних і слабких сторін;
- вибір і обґрунтування оптимальних інструментів і технологій для розробки платформи;
- розробка ключових компонентів системи, алгоритмів функціонування і користувацького інтерфейсу;
- впровадження, тестування та оптимізація веб-застосунку для забезпечення його надійності та продуктивності.

Об’єктом дослідження є система управління веб-застосунком, що дозволяє ефективно розміщувати, переглядати і шукати оголошення про продаж автомобілів.

Предметом дослідження є використання сучасних веб-орієнтованих технологій для розробки платформи оголошень про продаж автомобілів, яка забезпечує зручність користувачів, ефективність взаємодії між продавцями і покупцями, а також додаткові інтеграції для підвищення доступності сервісу. Особлива увага приділяється впровадженню таких технологій, що сприяють покращенню користувацького досвіду, швидкості роботи системи та надійності її функціонування. Це дозволяє створити сучасний веб-застосунок, який здатний адаптуватися під різні пристрої та забезпечувати якісну підтримку великої аудиторії користувачів.

У процесі розробки веб-застосунку застосовуються методи об’єктно-орієнтованого проектування, сучасного веб-програмування, розробки інтерфейсів користувача, а також технології взаємодії з базами даних і сторонніми API. Використання актуального технологічного стеку дозволяє досягти високого рівня продуктивності, масштабованості та безпеки системи, що є необхідною умовою для її практичного використання в умовах сучасного інформаційного середовища.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Постановка задачі

Розробка веб-сайту для розміщення оголошень про продаж автомобілів є важливим завданням, яке спрямоване на створення зручної та ефективної онлайн-платформи для купівлі-продажу автомобілів. Суть проєкту полягає у забезпеченні користувачів можливістю легко розміщувати, переглядати та керувати оголошеннями, а також комунікувати між собою для здійснення угод.

Основними елементами сайту є система реєстрації та авторизації, яка підтримує як стандартний метод входу, так і авторизацію через сторонні сервіси, зокрема Google [18]. Користувачі мають можливість створювати оголошення, редагувати їх, активовувати або деактивовувати, видаляти, а також додавати до списку порівняння чи обраного. На сайті реалізовано посторінковий каталог оголошень із функцією сортування та фільтрації, що дає змогу зручно переглядати детальну інформацію про кожний автомобіль. Для комунікації між продавцем і покупцем передбачено чат у режимі реального часу. Крім того, доступний функціонал відгуків, який дозволяє користувачам оцінювати продавців, що сприяє формуванню довіри. Адміністративна панель забезпечує управління користувачами, оголошеннями та модерацію контенту.

Весь цей функціонал в свою чергу не повинен допускати та пропускати помилки введення, нелогічні дії, спроби взлому. Дані мають надійно зберігатися та бути швидко доступні. Інтерфейс сайту має бути інтуїтивно зрозумілим та зручним для користування на різних типах пристроїв.

Для реалізації сайту необхідно виконати низку технічних завдань. Насамперед потрібно визначити стек технологій, який забезпечить ефективну розробку, підтримку та масштабованість проєкту. Важливо також обрати відповідний фреймворк, що дозволить прискорити процес створення сайту.

Наступним етапом є розробка структури бази даних. Вона повинна охоплювати інформацію про користувачів, оголошення, зображення,

повідомлення та відгуки. Одночасно потрібно передбачити надійні засоби безпеки, зокрема шифрування персональних даних.

Особливу увагу слід приділити користувацькому інтерфейсу. Його дизайн має бути інтуїтивно зрозумілим, адаптивним для різних типів пристроїв і доповнений інтерактивними елементами, що забезпечують комфортну взаємодію з платформою.

Функціональна частина сайту включає реалізацію механізмів реєстрації та авторизації користувачів, створення, редагування й перегляду оголошень, а також впровадження чат-системи для миттєвого обміну повідомленнями. Крім того, передбачено налаштування системи оцінювання та відгуків для покращення довіри між учасниками платформи.

Для управління контентом необхідно створити адміністративну панель, яка дозволить контролювати оголошення, користувачів і модерувати відгуки. Безпека системи повинна бути посилена захистом від можливих атак, таких як SQL-ін’єкції та XSS, а також впровадженням електронної верифікації користувачів.

Завершальним етапом є налаштування середовища для розгортання сайту та забезпечення його стабільної роботи. Підсумовуючи, основними завданнями є аналіз існуючих аналогів і їх можливостей, підготовка даних для наповнення системи, побудова структури бази даних, реалізація ролей користувачів із відповідними рівнями доступу, створення та налаштування адміністративної панелі, проєктування архітектури системи, а також комплексне тестування перед запуском.

Результатом реалізації проєкту стане сучасна онлайн-платформа для розміщення оголошень про продаж автомобілів, що відповідає актуальним потребам користувачів. Завдяки продуманій архітектурі, зручному інтерфейсу та широкому функціоналу, система забезпечуватиме комфортну взаємодію між продавцями й покупцями. Використання надійного технологічного стеку, ефективної логіки на сервері та захищених API гарантує стабільну, швидку й безпечну роботу всіх компонентів платформи.

1.2. Аналіз аналогів програмного продукту

Перш ніж розпочати розробку або вдосконалення програмного продукту, важливо провести аналіз існуючих рішень на ринку. Це дозволяє оцінити доступні можливості, визначити сильні та слабкі сторони конкурентів, а також знайти ідеї для покращення власного сервісу. Аналіз аналогів допомагає зрозуміти поточний стан галузі, виявити тенденції та сформувати конкурентоспроможну стратегію розвитку продукту.

Під час аналізу ринку сервісів для розміщення оголошень про продаж автомобілів було виокремлено три провідні платформи: AUTO.RIA, RST та OLX (частково, оскільки цей сервіс містить категорію оголошень для продажу авто). Нижче наведено детальний розгляд кожного з них, з акцентом на функціональні можливості, переваги та недоліки.

Сервіс AUTO.RIA [1]

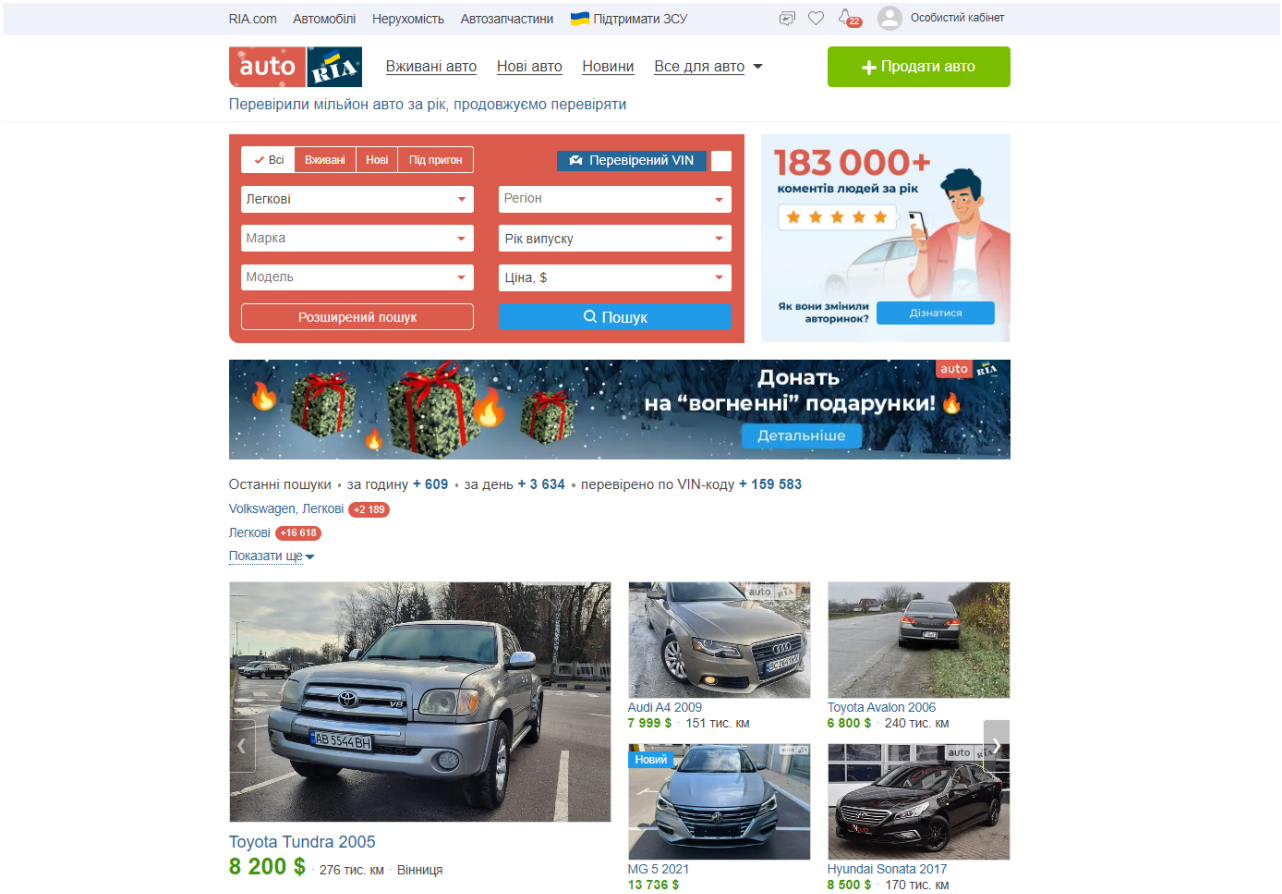


Рис. 1.1 – Вигляд головної сторінки веб-сайту AUTO.RIA

AUTO.RIA є найпопулярнішим онлайн-майданчиком для купівлі та продажу автомобілів в Україні (див. рис. 1.1). Платформа охоплює не лише

легкові авто, а й транспортні засоби інших категорій, зокрема автобуси, спецтехніку, вантажівки, водний і повітряний транспорт. Серед основних функціональних можливостей сервісу варто відзначити зручну систему фільтрації та сортування оголошень за такими параметрами, як ціна, рік випуску, пробіг, тип кузова тощо. Користувачі можуть ознайомитися з технічними характеристиками автомобіля, перевірити VIN-код і історію транспортного засобу. Платформа підтримує комунікацію між продавцем і покупцем через вбудований чат і можливість здійснення дзвінків безпосередньо через сайт. Для зручності користувачів реалізовано функції додавання оголошень до списку “Обраного” та порівняння автомобілів. Додатково передбачена опція платного підняття оголошень у рейтингу, перегляд відеопрезентацій авто та підписка на оновлення відповідно до вибраних параметрів. Для мобільних користувачів доступні офіційні додатки AUTO.RIA на платформах iOS та Android.

Переваги:

- велика аудиторія користувачів та висока довіра до сервісу;
- вузька спеціалізація на транспортних засобах, що забезпечує якісний підхід до ринку;
- глибока перевірка продавців та авто (перевірка VIN-коду, історії реєстрації тощо);
- багато безкоштовних функцій, зокрема можливість розміщення без реєстрації;
- потужна система фільтрів і сортування — дозволяє швидко знаходити потрібний автомобіль за великим переліком параметрів (модель, пробіг, коробка передач, стан, ціна тощо).

Недоліки:

- більшість просунутих функцій доступні лише за додаткову оплату;
- висока конкуренція серед продавців може ускладнювати продаж авто без додаткової реклами.

Сервіс RST [2]

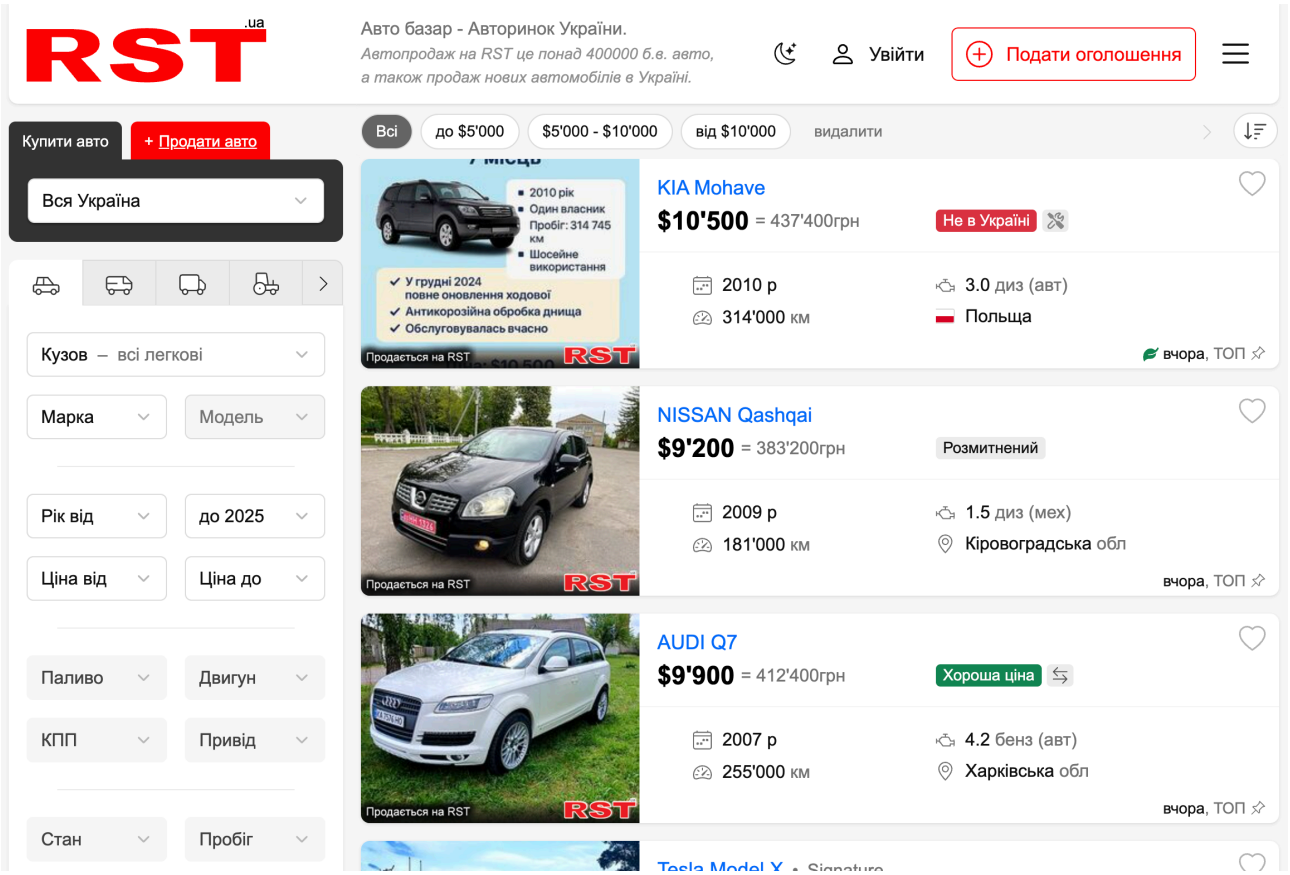


Рис. 1.2 – Вигляд головної сторінки веб-сайту RST

RST є одним із найстаріших сервісів оголошень про продаж автомобілів в Україні, який значно поступається сучасним конкурентам за зручністю користування та дизайном (див. рис. 1.2). Користувачі можуть безкоштовно розміщувати оголошення без необхідності реєстрації, а інтерфейс платформи вирізняється мінімалістичністю та простотою у використанні. Для пошуку автомобілів передбачена базова система фільтрації за основними параметрами. Водночас, платформа не має додаткових інструментів, таких як порівняння автомобілів або чат для комунікації між продавцем і покупцем.

Переваги:

- можливість розміщення оголошень без реєстрації;
- простота використання.

Недоліки:

- застарілий дизайн та відсутність мобільного додатка;
- відсутність чату та системи перевірки авто;

- немає можливості порівнювати кілька оголошень;
- менша популярність у порівнянні з AUTO.RIA, що знижує ефективність продажів.

Сервіс OLX [3]

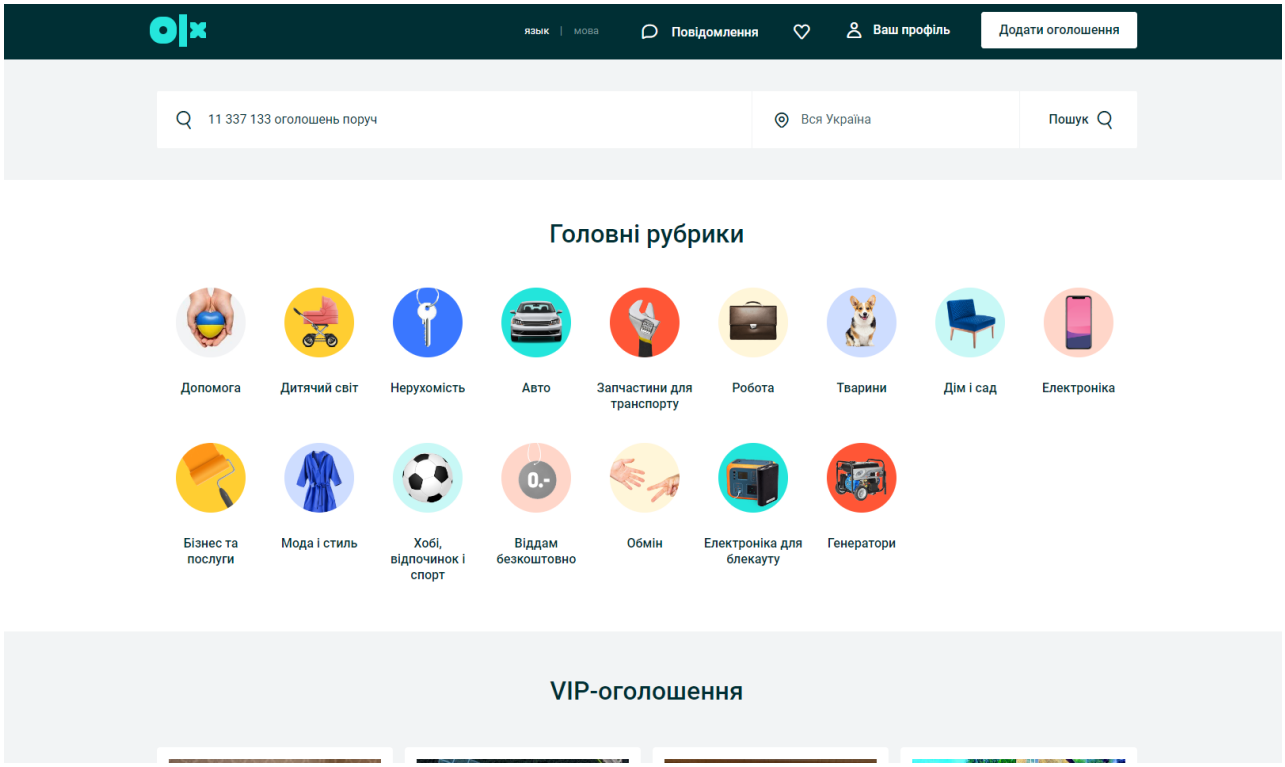


Рис. 1.3 – Вигляд головної сторінки веб-сайту OLX

OLX — це загальнопрофільний сайт оголошень, який включає окрему категорію для купівлі-продажу автомобілів (див. рис. 1.3). Платформа має велику базу оголошень не лише з автомобілів, а й з інших категорій товарів. Користувачі можуть спілкуватися з продавцями через чат та залишати відгуки про них. Для покращення видимості оголошень доступні вбудовані рекламні інструменти, що дозволяють піднімати оголошення у рейтингу. Крім того, платформа підтримує можливість додавання оголошень до списку “Обраного” для зручного подальшого перегляду.

Переваги:

- велика кількість користувачів;
- чат із продавцем безпосередньо на платформі;
- зручний мобільний додаток.

Недоліки:

- відсутність спеціалізації на автомобілях, що зменшує ефективність продажу в порівнянні з AUTO.RIA;
- оголошення можна розміщувати лише після реєстрації;
- відсутність функції порівняння оголошень;
- менш деталізована інформація про автомобілі у порівнянні з AUTO.RIA.

Таблиця 1.1

Порівняння аналогів

Функція / Сервіс	AUTO.RIA	RST	OLX	Мій
Спеціалізація на авто	+	+	-	+
Чат із продавцем	+	-	+	+
Фільтрація та сортування	+	+	+	+
Порівняння оголошень	+	-	-	+
Додавання в “Обране”	+	+	+	+
Можливість розміщення без реєстрації	+	+	-	-
Перевірка продавців та авто	+	-	-	+
Перегляд історії авто за VIN-кодом	+	-	-	+
Система рейтингу продавців	+	-	+	+
Телеграм-бот для публікацій	-	-	-	+

Серед трьох проаналізованих платформ найбільш функціональним і зручним сервісом для продажу автомобілів в Україні є AUTO.RIA. Ця платформа пропонує широкий спектр можливостей, які не лише спрощують процес пошуку найкращих пропозицій для покупців, а й сприяють швидкому та ефективному продажу транспортних засобів. Натомість сервіс RST, хоча й дозволяє розміщення оголошень без реєстрації, має суттєві обмеження у функціональності, що значно знижує його конкурентоспроможність на ринку.

Платформа OLX охоплює широку аудиторію завдяки своїй універсальності, однак через відсутність спеціалізації у сфері продажу автомобілів вона не здатна забезпечити такий рівень сервісу, як AUTO.RIA. Враховуючи проведений аналіз, при розробці власної системи доцільно орієнтуватися на функціонал AUTO.RIA, адаптуючи та впроваджуючи подібні рішення для підвищення ефективності й привабливості майбутнього сервісу, а також розширюючи його новими функціями для зручності користування платформою.

1.3. Вибір архітектури веб-орієнтованої системи

Вибір архітектури веб-орієнтованої системи є важливим етапом розробки, оскільки він впливає на продуктивність, масштабованість та зручність подальшого розвитку проєкту. Архітектура визначає, як різні компоненти взаємодіють між собою, як здійснюється передача даних та наскільки легко можна модифікувати або розширювати систему. Правильний вибір архітектури дозволяє мінімізувати ризики під час розробки та оптимізувати використання ресурсів, що особливо важливо при зростанні кількості користувачів і запитів.

У цьому проєкті було прийнято рішення використовувати RESTful API архітектуру, оскільки вона дозволяє стандартизувати взаємодію між клієнтською та серверною частинами, забезпечуючи гнучкість у розвитку та підтримці [22]. Даний підхід спрощує інтеграцію нових функцій, а також дає можливість легко масштабувати систему при збільшенні навантаження (рис. 1.4).

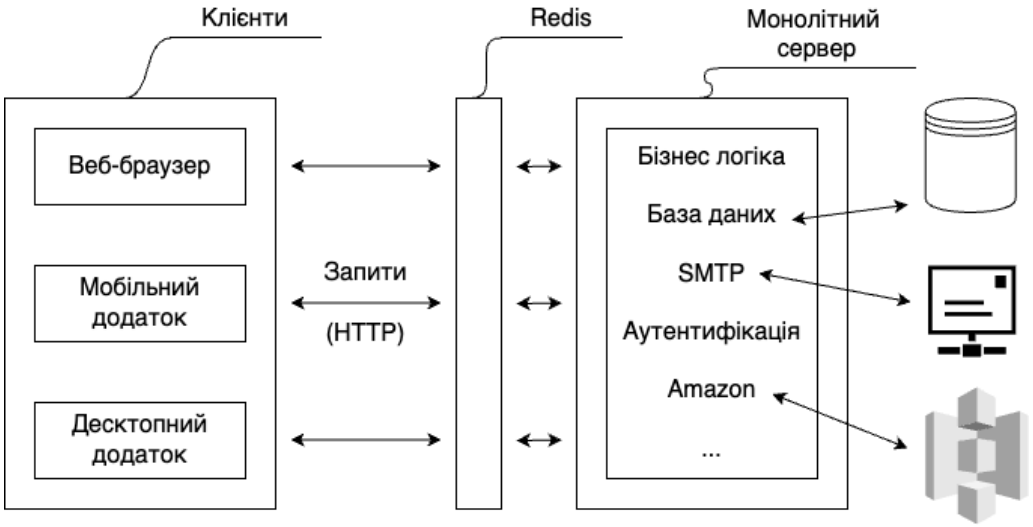


Рис. 1.4 – Клієнт-серверна архітектура в поєднання з монолітним сервером

RESTful API сприяє покращенню структурованості коду, забезпечує чітке розділення відповідальностей між компонентами системи та спрощує обмін даними між клієнтом і сервером, уніфікуючи формат запитів і відповідей, полегшуючи розробку та підтримку системи.

Серверну частину буде реалізовано на основі монолітної архітектури з використанням фреймворку Express із модульним поділом коду за функціональними областями [6]. Такий підхід є виправданим для даного проєкту, оскільки забезпечує просту та гнучку структуру, яка дозволяє швидко налаштовувати маршрути, middleware та логіку обробки запитів. Express дозволяє гнучко налаштувати власну структуру проєкту, що дає змогу організувати код максимально модульно та логічно та дає повний контроль над побудовою API, що сприяє гнучкій реалізації функціоналу, зокрема аутентифікації, авторизації, логування та обробки помилок. Монолітна архітектура зі спільною кодовою базою спрощує розробку, тестування та підтримку сервісу, а використання Express забезпечує високу продуктивність і стабільність при виконанні основних серверних задач.

Клієнтська частина буде реалізована за клієнт-серверною архітектурою і побудована з використанням бібліотеки React.js [16]. Вона відповідає за відображення інтерфейсу користувача та взаємодію з сервером через REST API за допомогою HTTP-запитів. Для ефективного управління асинхронними запитами та оптимізації кешування використовується бібліотека TanStack Query у поєднанні з Axios, що забезпечує зручне оновлення даних і повторне використання запитів [15].

Архітектурно клієнтська частина відповідатиме патерну MVVM (Model-View-ViewModel), де модель відповідає за отримання і збереження даних з сервера, що реалізовано через API-запити та TanStack Query, які забезпечують доступ до даних і їх кешування [13]. Логіка отримання, обробки і трансформації даних для користувацького інтерфейсу, а також управління станом асинхронних операцій і відображенням помилок інкапсульована у кастомних React-хуках, що виконують роль ViewModel [12]. Сам інтерфейс користувача реалізовано за

допомогою React-компонентів, які отримують підготовлені ViewModel дані і відповідають за візуальне відображення і обробку користувацьких подій (рис. 1.5).

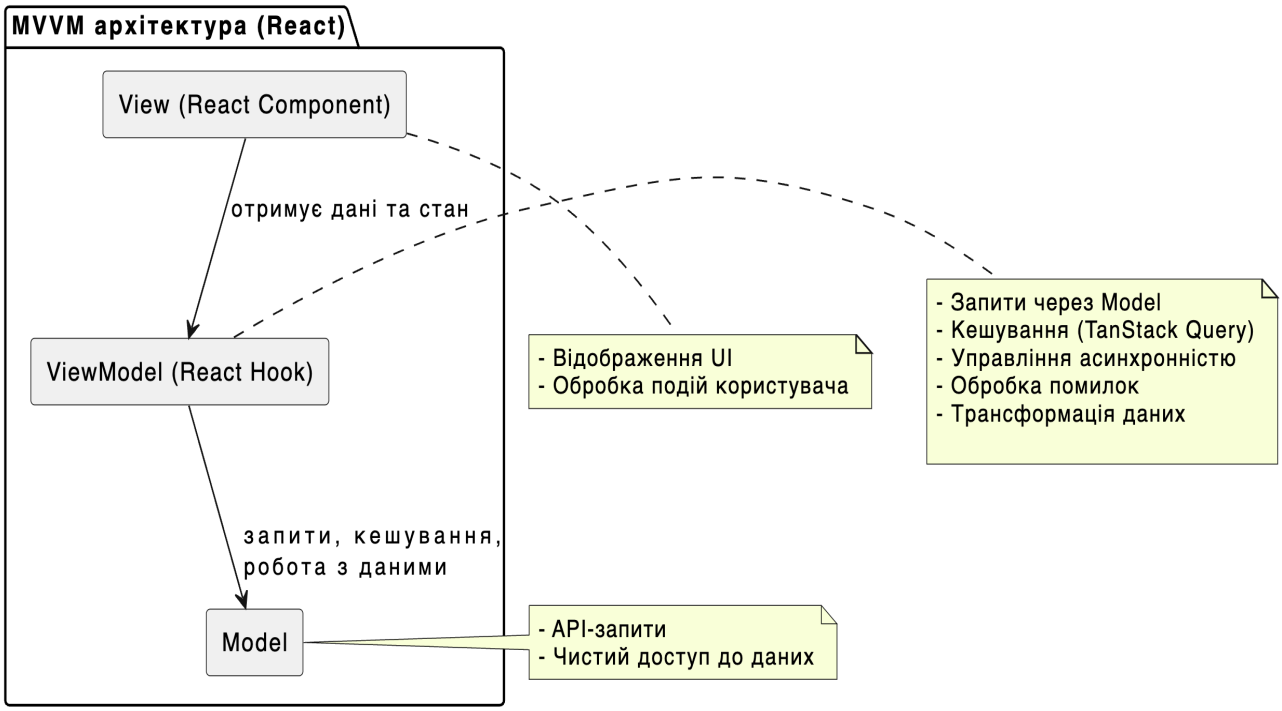


Рис. 1.5 – Візуалізація патерну MVVM в архітектурі React-застосунку

Такий розподіл обов'язків підвищує модульність, підтримуваність і тестованість клієнтської частини, розділяючи логіку бізнесу і управління станом від безпосереднього відображення інтерфейсу.

Для забезпечення швидкого доступу до даних і зниження навантаження на сервер використовується кешування за допомогою Redis, що суттєво підвищує продуктивність при обробці запитів [24]. Окрім цього, для реалізації функціоналу реального часу застосовується WebSocket-сервер на базі Socket.IO [21]. Такий підхід дозволяє встановлювати постійне двонаправлене з'єднання між клієнтом і сервером, що забезпечує миттєву передачу повідомлень без необхідності повторних HTTP-запитів. Це особливо важливо для чатів, оновлення даних у реальному часі та інших інтерактивних функцій. Завдяки цьому система стає більш чутливою до змін і підвищує рівень взаємодії користувачів, що є критичною перевагою для динамічних веб-застосунків.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для розробки серверної частини було обрано мову TypeScript, платформу Node.js та фреймворк Express. TypeScript забезпечує статичну типізацію, що дозволяє зменшити кількість помилок під час розробки та підвищує надійність коду [4]. Express, як мінімалістичний і гнучкий фреймворк, дає змогу самостійно визначати архітектуру проєкту, що дозволяє реалізувати модульну структуру з розділенням за доменними областями, спрощуючи масштабування та підтримку. Node.js забезпечує асинхронну обробку запитів і високу продуктивність, що робить його ефективним вибором для розробки серверної частини.

Клієнтська частина реалізована за допомогою React. Цей фреймворк був обраний через його компонентний підхід, що дозволяє ефективно керувати інтерфейсом, забезпечує перевикористання коду та спрощує розширення функціональності. Використання віртуального DOM дозволяє оптимізувати рендеринг сторінки, що позитивно впливає на продуктивність. Для стилізації інтерфейсу застосовано Tailwind CSS у поєднанні з компонентною бібліотекою shadcn/ui, що значно пришвидшує розробку та дозволяє створювати гнучкі й адаптивні UI-компоненти [26]. Це рішення забезпечує швидкість розробки, спрощує підтримку стилів та дозволяє легко змінювати зовнішній вигляд застосунку без глибоких правок у CSS-коді [36].

Для взаємодії з серверною частиною та виконання HTTP-запитів до API використовується бібліотека TanStack Query. Вона забезпечує ефективне управління асинхронними операціями та автоматичне кешування результатів запитів, що підвищує продуктивність застосунку. Бібліотека підтримує автоматичне оновлення даних у фоновому режимі, завдяки чому інформація залишається актуальною без додаткового втручання з боку розробника. Крім того, TanStack Query спрощує обробку станів завантаження, помилок і повторних спроб, що робить застосунок більш стабільним та покращує користувацький досвід. Особливу увагу заслуговує підтримка оптимістичних оновлень, яка

дозволяє миттєво відображати зміни в інтерфейсі без очікування відповіді сервера, роблячи взаємодію користувача більш швидкою.

Для зберігання даних у проєкті обрано реляційну базу даних PostgreSQL, що є одним із найнадійніших та найпотужніших рішень серед СКБД з відкритим вихідним кодом [7]. Вона забезпечує високу продуктивність, підтримку транзакцій, складних запитів і реляційних зв’язків, а також ефективну роботу з розширеними типами даних, зокрема масивами, JSONB і enum. Це особливо важливо при зберіганні та обробці структурованої інформації — користувачів, повідомлень, оголошень, автомобілів тощо. PostgreSQL не поступається комерційним аналогам за функціональністю, є безкоштовною, стабільною, широко підтримуваною спільнотою та має зручну інтеграцію з ORM TypeORM, що є додатковою перевагою для розробки на сучасному стеку.

Вибір саме реляційної СКБД зумовлений структурованою природою даних у цьому проєкті. Такі сутності, як користувачі, повідомлення, оголошення та транспортні засоби мають чітко визначені атрибути та взаємозв’язки, які зручно моделювати через таблиці з зовнішніми ключами. Реляційна модель забезпечує цілісність даних, дозволяє ефективно використовувати каскадні операції, індексацію та складні SQL-запити, що робить її ідеальною для платформи з великою кількістю пов’язаних об’єктів та логіки бізнес-процесів.

Для взаємодії з базою даних використовується ORM TypeORM, яка дозволяє працювати з даними у вигляді об’єктів, легко налаштовувати зв’язки між сутностями, автоматизувати створення та виконання міграцій, а також забезпечує типізацію на рівні моделей [8]. Завдяки цьому значно спрощується написання SQL-запитів, забезпечується узгодженість даних між кодом і структурою бази та підвищується гнучкість при зміні або розширенні схеми даних у процесі розробки. Такий підхід сприяє швидкій і стабільній розробці, особливо при роботі над масштабними проєктами з багатьма взаємопов’язаними таблицями.

Для оптимізації роботи сервера передбачено кешування за допомогою Redis, що дозволяє значно зменшити навантаження на базу даних і прискорити

обробку запитів [24]. Використання Redis є доцільним у випадках, коли необхідно швидко зчитувати часто використовувані дані, наприклад, результати запитів або сесійні дані.

Для збереження зображень у проєкті використовується сервіс AWS S3, який забезпечує надійне, масштабоване та безпечне зберігання файлів у хмарі [17]. Використання S3 дозволяє уникнути обмежень, пов’язаних із локальним зберіганням, таких як обмежений простір на сервері чи ризики втрати даних при збоях. Такий підхід оптимізує продуктивність системи, дозволяючи швидко обробляти запити на завантаження зображень, що особливо важливо для забезпечення комфортного користувацького досвіду.

З метою забезпечення зручності розгортання та підтримки всіх сервісів використовується контейнеризація через Docker [5]. Усі сервіси, включаючи серверну частину, клієнтську частину, базу даних, кеш та WebSocket-сервер, працюють у контейнерах, що забезпечує гнучкість у налаштуванні та можливість швидкого перенесення застосунку на інші середовища. Для управління контейнерами використовується docker-compose, що дозволяє автоматизувати процес запуску всіх компонентів, забезпечуючи зручне масштабування та централізоване керування сервісами.

Зважаючи на всі обрані інструменти та підходи, архітектура та технологічний стек веб-сайту забезпечують ефективність, продуктивність та зручність подальшого розвитку проєкту. Використання сучасних технологій дозволяє масштабувати систему, інтегрувати нові сервіси та забезпечити стабільну та безпечну роботу веб-застосунку навіть при високих навантаженнях.

Висновки до першого розділу

У першому розділі визначено основні завдання та технічні вимоги для розробки веб-системи. Проведено аналіз існуючих рішень із визначенням їх функціональності, переваг і недоліків на основі порівняння з аналогами. Архітектура системи обрана з урахуванням вимог проекту: серверна частина реалізована як модульний моноліт на базі Express, що забезпечує зручну

організацію коду та можливість масштабування, а клієнтська частина — за клієнт-серверною моделлю з використанням сучасних фронтенд-технологій. У клієнтській частині застосовується підхід MVVM для ефективного розділення логіки, даних та інтерфейсу, що полегшує підтримку і розвиток проекту. Для реалізації вибрано стек технологій: PostgreSQL як надійну систему управління базами даних для збереження складних зв’язків, React — для побудови інтерфейсу, TanStack Query — для роботи з API та обробки запитів, а серверна частина побудована на Node.js з використанням Express, що забезпечує високу продуктивність і гнучкість розробки.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ

2.1. Визначення варіантів використання та об’єктно-орієнтованої структури системи

Аналіз вимог охоплює визначення та уточнення як функціональних, так і нефункціональних характеристик програмного забезпечення [30]. Він включає оцінку часу реакції системи на дії користувачів, опис можливих сценаріїв її використання, а також створення загальної схеми елементів системи з поясненням їх структури, взаємодії та методів функціонування. Такий підхід дозволяє досягти чіткого розуміння потреб користувачів і технічних параметрів, необхідних для розробки зручного та ефективного програмного продукту.

Функціональні вимоги

Функціональні вимоги описують ключові завдання та можливості системи, які необхідно реалізувати для забезпечення її цільового призначення та відповідності очікуванням користувачів [31].

Передусім зазначимо, що в межах сервісу передбачено наявність таких ролей:

1. Адміністратор – користувач системи, наділений повними правами доступу до системи, статистичної інформації, наданням ролей.
2. Модератор – користувач системи, наділений правами для модерації оголошень, контролю порядку та дій користувачів.
3. Користувач сайту – користувач системи, наділений усіма правами звичайного користувача з ціллю продати або придбати автомобіль. Не аутентифікований користувач має обмежений набір доступних йому функцій.

Для неаутентифікованого користувача система надає можливість переглядати доступні оголошення, ознайомлюватися з детальною інформацією про кожне з них, включно з характеристиками автомобіля, фотографіями та контактними даними продавця. Також доступна функція пошуку й фільтрації автомобілів за визначеними критеріями. Окрім того, користувач може пройти

реєстрацію або авторизацію через електронну пошту й пароль або за допомогою зовнішніх сервісів через OAuth (наприклад, Google).

Після авторизації користувач отримує доступ до розширеного функціоналу, що дозволяє створювати нові оголошення про продаж автомобіля, редагувати та видаляти власні оголошення, а також змінювати їхній статус (активувати або деактивувати). Користувач може переглядати всі наявні оголошення, зберігати вибрані до списку обраного, додавати їх до списку для порівняння, спілкуватися з іншими користувачами через чат, залишати відгуки та виставляти оцінки продавцям. Крім того, доступна можливість редагування персональної інформації в обліковому записі.

Модератор має доступ до перегляду та модерації всіх оголошень, а також до видалення інформаційних блоків, розміщених на сайті. Він контролює дотримання правил користування платформою та забезпечує порядок контенту, який публікується іншими користувачами.

Адміністратор отримує найвищий рівень доступу. Він має можливість призначати ролі іншим користувачам, керувати модулями платформи, управляти обліковими записами користувачів, а також адмініструвати усіма оголошеннями.

Нефункціональні вимоги

Нефункціональні вимоги описують властивості та показники якості системи, що впливають на її ефективність, надійність і зручність використання [32]. Вони охоплюють кілька ключових аспектів.

Однією з основних вимог є сприйняття та зручність використання. Інтерфейс користувача має бути інтуїтивно зрозумілим, не вимагати спеціальної підготовки від користувачів та бути адаптованим до роботи на мобільних пристроях, що забезпечує доступність і комфорт у взаємодії з системою.

Продуктивність і масштабованість також є критично важливими. Система повинна забезпечувати час відгуку не більше 5 секунд для основних операцій і до 15 секунд для складних запитів. Водночас вона має підтримувати стабільну роботу за умови одночасного використання щонайменше 5000 користувачами без помітного зниження швидкодії.

Ще одним важливим аспектом є безпека. Усі дані користувачів повинні передаватися через захищені канали, а паролі зберігатися у вигляді хешів з використанням сучасних криптографічних алгоритмів. Крім того, програмне забезпечення має регулярно оновлюватися з метою усунення потенційних вразливостей.

Останнім компонентом є сумісність. Система повинна коректно працювати в усіх основних сучасних веб-браузерах, зокрема Google Chrome, Mozilla Firefox, Safari та Microsoft Edge, а також підтримувати різні операційні системи, що гарантує широку доступність і зручність для користувачів із різними технічними платформами.

Вимоги до часових характеристик реакції системи на дії користувача

Часові показники реакції системи виступають важливим критерієм її продуктивності та рівня якості обслуговування. Для забезпечення зручного користування сайтом розміщення оголошень про продаж автомобілів встановлюються наступні вимоги:

1. Загальна доступність системи: час, витрачений на технічне обслуговування, не повинен перевищувати 3% від загального часу роботи системи на місяць.
2. Час розміщення оголошення: повний процес створення нового оголошення (заповнення форми, додавання фото, публікація) не повинен перевищувати 5 хвилин для пересічного користувача.
3. Час відгуку на типові дії користувача:
   - перехід між сторінками сайту (категорії, список оголошень, деталі автомобіля) — не більше 3 секунд;
   - пошук автомобілів за фільтрами або ключовими словами — не більше 5 секунд;
   - завантаження галереї зображень автомобіля — не більше 5 секунд.
4. Час відгуку на складні дії користувача: обробка складних запитів (наприклад, багатофакторна фільтрація за ціною, пробігом, роком випуску тощо) — не більше 10 секунд.

5. Час обробки запитів до бази даних: час очікування відповіді на один запит до бази даних не повинен перевищувати 10 секунд.
6. Час авторизації та реєстрації користувачів: процеси реєстрації нового користувача або авторизації існуючого повинні завершуватися протягом 3 секунд.
7. Час реагування адміністративних функцій: перегляд логів, модерація оголошень або зміна статусів облікових записів — не більше 5 секунд.
8. Час відображення повідомлень користувачу: системні повідомлення (про успішне розміщення оголошення, зміну пароля, помилки введення даних) повинні відображатися не пізніше ніж через 2 секунди після обробки відповідної дії.

Специфікація варіантів використання системи

Специфікація варіантів використання системи є важливою складовою процесу проєктування, оскільки вона описує основні сценарії взаємодії користувачів із сервісом та допомагає зрозуміти, як саме система підтримує їхні дії для досягнення цілей. Вона дає змогу виявити та задокументувати всі можливі дії користувачів, визначити ключові функціональні можливості системи та впевнитися, що вони будуть враховані під час розробки.

Для кращого візуального представлення цих сценаріїв створюється діаграма варіантів використання, яка наочно демонструє структуру взаємодії користувачів із системою (див. Додаток А). Діаграма сприяє глибшому розумінню функціональності сервісу всіма учасниками.

Нижче наведено основні сценарії, що ілюструють типові випадки використання сайту для розміщення оголошень про продаж автомобілів.

Таблиця 2.1

Варіанти використання системи

Основний актор	Найменування	Формулювання
Усі актори	Авторизація	Кожен користувач системи (окрім не зареєстрованого користувача) має можливість здійснити вхід на сайт, використовуючи логін та пароль або OAuth (Google)

Продовження таблиці 2.1

Адміністратор	Надання ролей	Цей варіант використання дозволяє адміністратору надавати користувачам нові ролі
Адміністратор, модератор	Модерація оголошень	Цей варіант використання дозволяє адміністратору та модератору проводити перевірку оголошень перед їх публікацією
Адміністратор	Контроль роботи сайту	Цей варіант використання дозволяє адміністратору контролювати дії, які відбуваються на сайті та реагувати на них
Адміністратор	Керування користувачами	Адміністратор має доступ до всіх користувачів системи та може переглядати інформацію про них, призначати ролі або блокувати їх
Адміністратор	Керування оголошеннями	Адміністратор має доступ до всіх оголошень платформи та може керувати ними
Модератор	Контроль дотримання правил користування сайтом	Модератор має можливість контролювати дотримання правил користування сайтом іншими користувачами
Модератор	Оперативне втручання в разі недотримання правил користування	Цей варіант використання дозволяє модератору оперативно реагувати на порушення та застосовувати відповідні санкції
Користувач сайту	Реєстрація на сайті	Цей варіант використання дозволяє користувачу зареєструватись на сайті для зручності подальшої взаємодії з ним
Користувач сайту	Розміщення та контроль оголошень	Цей варіант використання дозволяє користувачам сайту розміщувати оголошення, контролювати та керувати ними
Користувач сайту	Збереження оголошень до обраного	Користувач сайту має змогу зберегти певне оголошення до обраного
Користувач сайту	Додавання оголошень до порівняння	Цей варіант використання дозволяє користувачам сайту додавати певні оголошення до порівняння
Користувач сайту	Залишення відгуків та оцінювання продавців	Кожен користувач має можливість залишити відгук, а також оцінити продавця автомобіля
Користувач сайту	Комунікація між продавцем і покупцем	Цей варіант використання дозволяє продавцю і його клієнту зручно комунікувати, відстежувати історії переговорів
Користувач сайту	Розширений пошук автомобілів	Кожен користувач для зручності може використовувати різні параметри сортування і фільтрації для пошуку автомобіля

Таким чином, діаграма варіантів використання є важливою частиною процесу проєктування системи. Вона сприяє глибшому розумінню потреб користувачів. Завдяки аналізу різних сценаріїв взаємодії користувачів із системою, діаграма допомагає визначити основні функціональні можливості, які повинні бути реалізовані. Вона фокусує увагу на реальних очікуваннях і потребах користувачів, що дозволяє уникнути помилок і забезпечити відповідність функціоналу системи їхнім вимогам.

Об’єктно-орієнтована структура системи

Архітектура серверної частини застосунку побудована за модульним принципом, що забезпечує гнучкість, масштабованість та легкість у підтримці й розширенні системи. Кожен функціональний блок винесений в окремий модуль, який містить у собі пов’язані компоненти: контролери, сервіси, сутності, DTO, роутери та інші допоміжні елементи. Це дозволяє інкапсулювати логіку кожного домену в межах модуля, водночас відкриваючи необхідні функціональності для взаємодії з іншими модулями через надання доступу до потрібних функціональних частин між модулями.

Для відображення структури системи та взаємодії модулів буде використана діаграма класів. Діаграма класів є статичною структурною схемою, яка застосовується в об'єктно-орієнтованому програмуванні та моделюванні систем з метою візуалізації класів, їхніх атрибутів, методів і взаємозв'язків між ними [27].

Отже, для розробки програмного забезпечення та демонстрації взаємодії модулів системи було створено попередню об’єктно-орієнтовану модель системи у вигляді діаграми класів (див. Додаток Б).

Кожен модуль має свою внутрішню структуру і реалізує специфічну бізнес-логіку. Розглянемо їх детальніше.

Модуль account — відповідає за всю бізнес-логіку, пов’язану з управлінням користувачами: реєстрацією, авторизацією, Google OAuth, скиданням та відновленням паролів, а також обробкою пов’язаних операцій. Реалізує безпеку

доступу та валідацію вхідних даних. Для роботи використовує функціонал модулів aws, user, token та image.

Модуль additional-option — забезпечує зберігання додаткових характеристик і опцій автомобілів у спеціальній сутності. Відповідає за CRUD-операції з цими додатковими опціями та їх коректне використання у зв’язку з іншими модулями, що працюють з авто.

Модуль aws — інкапсулює всю роботу із сервісами AWS: зберігання, отримання, видалення файлів та інших даних. Забезпечує безпечний та оптимізований доступ до хмарного сховища, використовується іншими модулями для роботи із файлами, зображеннями, медіа тощо.

Модуль car — реалізує повний цикл роботи із сутністю автомобіля: створення, редагування, збереження характеристик та атрибутів авто. Модуль інтегрується з API для розпізнавання номерних знаків автомобілів на основі фото, що дозволяє автоматизувати введення даних [33]. Ця сутність логічно пов’язана із car-ad, проте зберігається окремо в базі даних. Модуль містить необхідні сервіси, DTO, контролери для повноцінної роботи з авто.

Модуль car-ad — відповідає за логіку роботи з оголошеннями про продаж автомобілів. Забезпечує публікацію, редагування, пошук і фільтрацію оголошень. Для своєї роботи взаємодіє з багатьма іншими модулями: user, aws, car, car-comparison, car-image, favorite-ad, additional-option та deleted-ad.

Модуль car-brand — управляє даними про бренди автомобілів. Використовує у своїй роботі модуль car-query для отримання актуальної інформації про бренди, моделі, їх характеристики і доступність, що спрощує підтримку та оновлення списків.

Модуль car-comparison — реалізує функціонал порівняння різних автомобілів за характеристиками, допомагаючи користувачам приймати обґрунтовані рішення. Взаємодіє з модулями car та aws.

Модуль car-image — містить бізнес-логіку, що пов’язана з роботою з зображеннями автомобілів. Для зберігання та обробки зображень використовує модуль image.

Модуль car-model — відповідає за управління моделями автомобілів. Під капотом активно використовує модуль car-query для отримання інформації про доступні моделі, їх характеристики та зв’язок із брендами.

Модуль car-query — реалізує інтеграцію із зовнішнім API, яке надає інформацію про марки автомобілів, моделі, покоління, залежно від року випуску та інших параметрів [34]. Цей модуль забезпечує достовірність і актуальність даних, що використовуються в car-brand, car-model та інших модулях.

Модуль conversation — відповідає за логіку системи переписок між користувачами (чатів). Забезпечує створення та ведення розмов, зберігання повідомлень, управління учасниками. Для роботи інтегрується з модулями user, aws, message.

Модуль deleted-ad — відповідає за обробку видалених оголошень. Використовується для збереження історії продажів автомобілів, що дає можливість відслідковувати повторні продажі і формувати історію автомобіля на платформі.

Модуль email — забезпечує функціонал відправки електронних листів користувачам, включаючи підтвердження реєстрації, скидання паролів, сповіщення тощо. Інкапсулює логіку формування та відправлення листів [23].

Модуль favorite-ad — реалізує функціонал роботи з обраними оголошеннями, дозволяючи користувачам зберігати, переглядати та керувати списком обраних авто. Взаємодіє з модулями aws та car-ad.

Модуль image — відповідає за загальну обробку зображень в системі, включаючи конвертацію форматів, стиснення, оптимізацію, перевірку на відповідність стандартам. Для зберігання файлів використовує модуль aws.

Модуль message — керує логікою повідомлень у системі, включно з обробкою, зберіганням та відправленням повідомлень. Для роботи інтегрується з модулями user, aws та conversation.

Модуль token — містить логіку роботи з JWT токенами для аутентифікації та авторизації користувачів, зокрема відповідальний за збереження і оновлення refresh токенів у базі даних, HTTPOnly cookie [25].

Модуль user — реалізує бізнес-логіку користувача, управління профілями, ролями та доступами. Для роботи використовує модуль aws.

Модуль user-review — відповідає за систему відгуків та рейтингів користувачів, дозволяє створювати, редагувати та переглядати відгуки. Взаємодіє з модулем aws.

Ця структура дозволяє побудувати систему для роботи з автомобільними оголошеннями, включаючи їх порівняння, переписки між користувачами та управління медіафайлами.

2.2. Розробка бази даних системи

Для зберігання всієї інформації, що обробляється сервісом, використовується PostgreSQL — одна з найнадійніших і найпотужніших систем керування базами даних з відкритим вихідним кодом [7]. Вона забезпечує високу продуктивність, підтримку транзакцій, складних запитів і реляційних зв’язків, а також ефективну роботу з розширеними типами даних. Це особливо важливо під час зберігання та обробки структурованої інформації: користувачів, повідомлень, оголошень, автомобілів тощо.

PostgreSQL не поступається комерційним аналогам за функціональністю, є безкоштовною, стабільною, активно підтримується спільнотою, а також має зручну інтеграцію з ORM TypeORM, що є суттєвою перевагою для розробки на сучасному технологічному стеку.

У проєкті сайту з розміщення оголошень про продаж автомобілів PostgreSQL використовується для зберігання складної та зв’язаної моделі даних: облікових записів користувачів, оголошень, автомобілів, обраного, порівнянь, чатів, повідомлень, зображень.

Жорсткі обмеження цілісності (foreign key, not null, unique) гарантують консистентність даних, а індекси та матеріалізовані подання — ефективну роботу аналітичних запитів. У результаті платформа отримує надійну, продуктивну та масштабовану основу для збирання і обробки даних.

Для сайту з розміщення оголошень про продаж автомобілів було спроєктовано 14 реляційних таблиць, які повністю покривають потреби зберігання даних платформи. Ці таблиці забезпечують зручне та структуроване збереження всієї необхідної інформації: additional_option, car, car_ad, car_additional_options_additional_option, car_comparison, car_image, conversation, deleted_ad, favorite_ad, Image, message, token, user, user_review.

Ця структура дозволяє ефективно працювати з пов'язаними даними, забезпечує масштабованість і підтримку всього необхідного функціоналу сервісу.

Таблиця 2.2

Опис таблиці user

Поле	Тип	Призначення
id	uuid	Унікальний ідентифікатор користувача
email	varchar(255)	Email користувача (унікальний)
name	varchar(255)	Ім'я користувача
surname	varchar(255)	Прізвище користувача
password	varchar(255)	Пароль користувача (може бути відсутній, якщо реєстрація через OAuth)
phone	varchar(255)	Номер телефону (унікальний, може бути відсутній)
isVerified	boolean	Статус підтвердження електронної пошти
token	Token (relation)	Токен авторизації, зв'язок один-до-одного
role	enum	Роль користувача (user, admin, moderator тощо)
image	Image (relation)	Зображення профілю користувача (необов'язкове, зв'язок один-до-одного)
createdAt	timestamp	Дата створення облікового запису
lastVisitAt	timestamp	Дата останнього візиту
isBlocked	boolean	Статус блокування користувача

User — основна таблиця користувачів, яка має зв'язок один-до-одного з таблицею Token для зберігання токенів авторизації, а також один-до-одного з таблицею Image для зберігання аватарок користувачів. Користувачі можуть бути пов'язані з багатьма оголошеннями через зв'язок багато-до-одного. Також користувачі беруть участь у відгуках, де зв'язок багато-до-одного вказує хто залишив відгук та хто його отримав. Крім того, користувачі можуть мати улюблені оголошення і порівнювати автомобілі через зв'язки багато-до-одного.

Таблиця 2.3

Опис таблиці user_review

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор відгуку
title	varchar(255)	Заголовок відгуку
text	text	Текст відгуку
userFromId	integer (FK)	Ідентифікатор користувача, який залишив відгук
userToId	integer (FK)	Ідентифікатор користувача, якому залишили відгук
dateOfCreation	timestamp	Дата створення відгуку
rate	integer	Оцінка (рейтинг)

UserReview — містить відгуки користувачів, кожен запис пов’язаний з користувачем, який залишив відгук, та користувачем, якому цей відгук належить, через зв’язки багато-до-одного.

Таблиця 2.4

Опис таблиці token

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор токена
refreshToken	varchar(255)	Токен оновлення (refresh token)
userId	integer (FK)	Зовнішній ключ на користувача (User)

Token — таблиця, що зберігає рефреш токени для користувачів, має зв’язок один-до-одного з таблицею User.

Таблиця 2.5

Опис таблиці message

Поле	Тип	Призначення
id	uuid	Унікальний ідентифікатор повідомлення
conversationId	uuid (FK)	Зовнішній ключ на розмову (Conversation)
senderId	uuid (FK)	Зовнішній ключ на користувача, який надіслав повідомлення
text	varchar(255)	Текст повідомлення
dateOfCreation	timestamp	Дата і час створення повідомлення

Message — таблиця повідомлень у чатах, пов’язана зі своїм діалогом через зв’язок багато-до-одного, а також зі своїм відправником теж через багато-до-одного.

Таблиця 2.6

Опис таблиці image

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор зображення
name	varchar(255)	Назва або шлях до файлу зображення

Image — таблиця зображень, використовується як в профілях користувачів, так і в автомобілях, зокрема в сутності CarImage.

Таблиця 2.7

Опис таблиці favorite_ad

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор запису
userId	integer (FK)	Зовнішній ключ на користувача
carAdId	integer (FK)	Зовнішній ключ на оголошення про автомобіль

FavoriteAd — таблиця, що зберігає улюблені оголошення користувачів, має зв’язки багато-до-одного з користувачами і оголошеннями автомобілів.

Таблиця 2.8

Опис таблиці deleted_ad

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор запису
mileage	integer	Пробіг автомобіля
vinCode	varchar(17)	VIN-код автомобіля (унікальний і проіндексований)
plateNumber	varchar(8), null	Номерний знак (може бути відсутній), проіндексований
deletedAt	timestamp	Дата видалення оголошення
userId	integer (FK)	Зовнішній ключ на користувача, який видалив оголошення

DeletedAd — зберігає інформацію про видалені оголошення, пов’язана багато-до-одного з користувачем, який видалив оголошення.

Таблиця 2.9

Опис таблиці conversation

Поле	Тип	Призначення
id	uuid	Унікальний ідентифікатор розмови
firstMemberId	uuid (FK)	Зовнішній ключ на першого учасника розмови (користувача)
secondMemberId	uuid (FK)	Зовнішній ключ на другого учасника розмови (користувача)

Продовження таблиці 2.9

lastFirstMemberVisit	timestamp, nullable	Дата останнього візиту першого учасника
lastSecondMemberVisit	timestamp, nullable	Дата останнього візиту другого учасника

Conversation — діалоги між двома користувачами, зберігає зв'язки багато-до-одного з двома учасниками, а також один-до-багатьох з повідомленнями.

Таблиця 2.10

Опис таблиці car_image

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор запису
imageId	integer (FK)	Зовнішній ключ на зображення (Image)
carId	integer (FK)	Зовнішній ключ на автомобіль (Car)
isMain	boolean	Прапорець основного зображення автомобіля

CarImage — пов'язує зображення з автомобілями через зв'язок один-до-одного з Image і багато-до-одного з Car.

Таблиця 2.11

Опис таблиці car_comparison

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор запису
userId	integer (FK)	Зовнішній ключ на користувача
carAdId	integer (FK)	Зовнішній ключ на оголошення про автомобіль

CarComparison — зберігає порівняння оголошень користувачами, має багато-до-одного зв'язок з користувачем і оголошенням.

Таблиця 2.12

Опис таблиці car_ad

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор оголошення
carId	integer (FK)	Зовнішній ключ на автомобіль
title	text	Заголовок оголошення
text	text	Опис оголошення

Продовження таблиці 2.12

dateOfCreation	timestamp	Дата створення оголошення
isActive	boolean	Статус активності оголошення (активне / неактивне)
userId	integer (FK)	Зовнішній ключ на користувача, який розмістив оголошення
status	enum	Статус оголошення (pending, approved, rejected тощо)
rejectedReason	text, nullable	Причина відхилення оголошення (якщо статус rejected)

CarAd — оголошення про продаж авто, пов’язане один-до-одного з автомобілем, має багато-до-одного зв’язок з користувачем, що розмістив оголошення, а також містить enum-статус для стану оголошення.

Таблиця 2.13

Опис таблиці car

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор автомобіля
carBrandId	text	Ідентифікатор бренду автомобіля
carModelId	text	Ідентифікатор моделі автомобіля
yearOfProduction	integer	Рік випуску
engineCapacity	real	Об'єм двигуна (літри)
fuel	enum (Fuel)	Тип палива (дизель, бензин тощо)
color	enum (Color)	Колір автомобіля
transmission	enum (Transmission)	Тип коробки передач
region	enum (Region)	Регіон, де знаходиться автомобіль
city	text	Місто розташування автомобіля
price	integer	Ціна автомобіля
wheelDrive	enum (WheelDrive)	Тип приводу коліс (передній, задній, повний)
numberOfSeats	integer	Кількість місць в автомобілі
mileage	integer	Пробіг автомобіля
vinCode	varchar(17), унікальний	VIN-код автомобіля (унікальний ідентифікатор)
plateNumber	varchar(8), nullable	Номерний знак автомобіля (може бути відсутнім)
carImages	one-to-many (CarImage)	Зображення автомобіля
additionalOptions	many-to-many (AdditionalOption)	Додаткові опції (наприклад, клімат-контроль, підігрів сидінь)

Car — центральна таблиця автомобілів з багатьма властивостями, має один-до-багатьох зв’язок з CarImage і багато-до-багатьох з таблицею додаткових опцій.

Таблиця 2.14

Опис таблиці additional_option

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор опції
name	varchar (string)	Назва додаткової опції
cars	many-to-many (Car)	Автомобілі, які мають цю опцію

AdditionalOption — додаткові опції автомобілів, пов’язані багато-до-багатьох з таблицею автомобілів.

Особливо варто звернути увагу на таблицю, яку TypeORM автоматично створює для реалізації зв’язку багато-до-багатьох між Car і AdditionalOption — це таблиця car_additional_options_additional_option. Вона містить лише два поля: зовнішні ключі на Car і на AdditionalOption, і служить для підтримки цього зв’язку. Ця проміжна таблиця не має власних полів, а виконує роль моста між двома сутностями у реляційній базі даних.

2.3. Проектування та реалізація алгоритмів роботи системи

Алгоритми мають ключове значення для забезпечення ефективної та коректної роботи системи, сприяючи вирішенню різноманітних завдань — від обробки запитів користувачів до управління потоками даних та їхньої обробки. Для моделювання алгоритмів роботи системи та відображення основних етапів взаємодії користувача з платформою для розміщення оголошень про продаж автомобілів були створені діаграми активності, які дозволяють детально описати послідовність дій у системі та ілюструють, як система реагує на різні дії користувачі [27]. Діаграма активності є потужним інструментом для відображення послідовності операцій, що відбуваються в межах бізнес-процесів. Вона показує, як різні дії взаємодіють між собою та в якому порядку вони виконуються. Це особливо корисно для оцінки ефективності роботи системи,

виявлення можливих вузьких місць або етапів, де можна оптимізувати взаємодію з користувачем.

Розглянемо діаграму активності, яка ілюструє дії користувача при першому вході користувача на сайт. Опція використання веб-сайту в режимі гостя є доступною, проте набір дій при цьому досить обмежений. Тому, щоб повноцінно мати змогу використовувати всі функції, користувачу запропоновано пройти процеси реєстрації та авторизації на сайті. Діаграма активності показує процес аутентифікації та реєстрації в системі, від введення користувачем даних до успішного входу в систему (рис. 2.1).

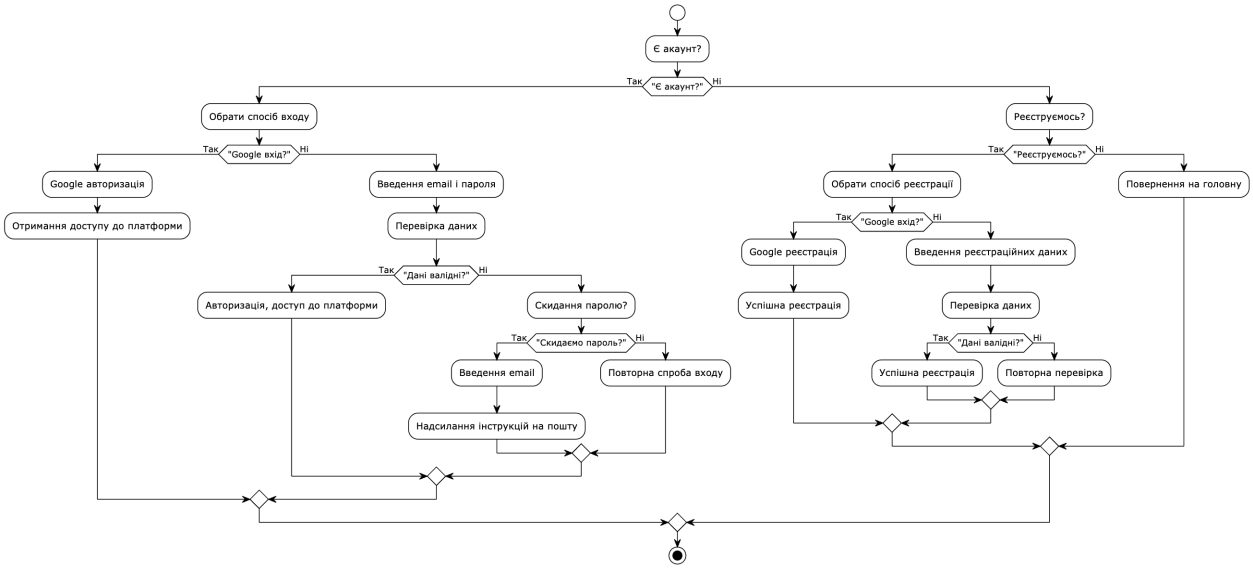


Рис. 2.1 – Діаграма активності при першому вході на сайт

Користувач починає взаємодію з системою, обираючи одну з доступних дій — авторизацію або реєстрацію. Якщо обрано авторизацію, користувач має змогу ввести свої дані для входу вручну або скористатись входом через Google. Після введення інформації система виконує перевірку: у разі валідних даних відбувається автентифікація та надається доступ до платформи, тоді як при помилці користувач повертається до форми авторизації для повторної спроби.

Крім того, у випадку, якщо користувач забув пароль, система дозволяє ініціювати процес скидання пароля. Для цього необхідно вказати адресу електронної пошти, на яку буде надіслано інструкції з відновлення доступу.

Якщо ж обрано реєстрацію, відкривається відповідна форма для введення даних. Користувач може пройти реєстрацію традиційним способом або

скористатись входом через Google. Після заповнення даних виконується перевірка. Якщо інформація містить помилки, користувач отримує повідомлення та можливість виправити їх. Якщо все введено коректно, реєстрація завершується успішно. У разі, якщо користувач вирішує скасувати реєстрацію, він повертається на головну сторінку.

Усі ці дії повторюються циклічно до моменту успішного завершення процесу входу або реєстрації.

Діаграма активності, зображена на рис. 2.2 демонструє ключову послідовність дій користувача, пов’язаних з управлінням оголошеннями в системі. Процес починається зі створення нового оголошення. Після його створення користувач має змогу редагувати вміст оголошення, видалити його або змінити статус — активувати чи деактивувати.

Крім створення, користувач також може переглядати всі доступні оголошення, застосовуючи фільтри та параметри сортування для зручного пошуку. У разі вибору конкретного оголошення система перевіряє, чи належить воно поточному користувачу. Якщо так — відкривається можливість редагувати або видалити оголошення. Якщо ж оголошення належить іншому користувачу, доступні інші дії: додавання оголошення до списку обраного, включення до списку для порівняння, ініціювання чату з продавцем, а також можливість залишити відгук про нього.

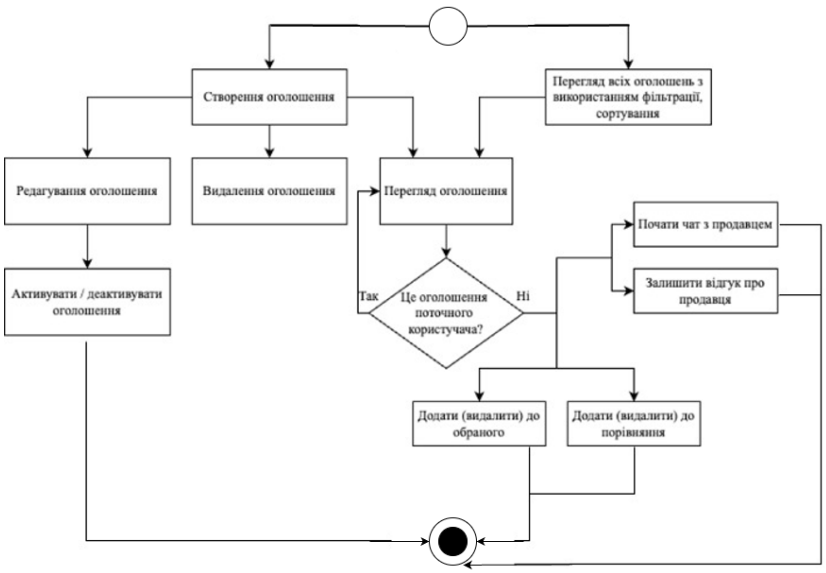


Рис. 2.2 – Діаграма активності дій з оголошеннями

Таким чином, діаграма відображає основну логіку взаємодії користувача з оголошеннями та варіанти подальших дій, що залежать від статусу володіння конкретним оголошенням.

2.4. Реалізація функціоналу системи

Перед початком розробки потрібно спроектувати базу даних. Це зручно зробити за допомогою графічних засобів, таких як сервіс draw.io [37]. У візуальному редакторі створюємо макет майбутньої бази даних: визначаємо основні таблиці, їх поля, типи даних та налаштовуємо зв'язки між сутностями (один до одного, один до багатьох, багато до багатьох).

На основі створеного макету бази даних реалізовуємо відповідну кодову структуру, використовуючи TypeORM — ORM-бібліотеку для роботи з базами даних у середовищі Node.js [8]. Кожна таблиця в макеті відповідає окремій сутності у коді. Для цього створюємо класи, які описують структуру таблиць, їхні поля, типи, первинні ключі, зв'язки та інші обмеження.

Наприклад, для збереження інформації про користувачів реалізуємо таку сутність:

```
@Entity()
export class User {
  @PrimaryGeneratedColumn("uuid")
  id: string;
  @Column({ unique: true })
  email: string;
  @Column()
  name: string;
  @Column()
  surname: string;
  @Column({ type: "character varying", nullable: true })
  password: string | null;
  @Column({ type: "character varying", unique: true, nullable: true })
  phone: string | null;
  @Column({ default: false })
  isVerified: boolean;
  @OneToOne(() => Token, (token) => token.user)
  token: Token;
  @Column({ type: "enum", enum: ["admin", "moderator", "user"], default: "user" })
  role: UserRole;
  @OneToOne(() => Image, { nullable: true })
  @JoinColumn()
  image: Image | null;
  @Column({ type: "timestamp", nullable: true })
  createdAt: Date | null;
```

```
  @Column({ type: "timestamp", nullable: true })
  lastVisitAt: Date | null;
  @Column({ type: "boolean", nullable: true, default: false })
  isBlocked: boolean | null;
}
```

Для збереження зображень у межах проєкту було прийнято рішення не використовувати ту ж машину, на якій розгорнуто веб-сервер. Такий підхід може спричинити надмірне навантаження на локальний диск, ускладнити резервне копіювання даних, а також створити проблеми у випадку масштабування системи чи зміни інфраструктури. Натомість було обрано надійне хмарне рішення — Amazon Web Services (AWS) S3. Цей сервіс призначений для зберігання об’єктів і чудово підходить для роботи з файлами різного типу, включно із зображеннями.

Після створення облікового запису в AWS відбувається налаштування сховища. У сервісі S3 створюється окремий Bucket, у якому зберігатимуться файли. Для цього задаються основні параметри, зокрема регіон розміщення та права доступу. Щоб веб-додаток міг безпечно передавати зображення до цього Bucket'a, у системі IAM створюється окремий користувач із чітко визначеними правами. Після цього генеруються унікальні ключі доступу, які зберігаються на сервері у змінних середовища. Це дозволяє інтегрувати AWS S3 у серверну логіку без ризику витоку чутливих даних.

Після налаштування доступу до AWS S3 логіку взаємодії з цим сервісом було інкапсульовано в окремий сервіс на сервері. Він базується на офіційних клієнтах AWS SDK та використовує кешування посилань на зображення за допомогою Redis для підвищення продуктивності та зменшення навантаження на S3. Такий підхід дозволяє централізовано керувати всіма операціями, пов’язаними із завантаженням, отриманням, оновленням і видаленням файлів у хмарному сховищі. Завдяки цьому код залишається чистим, зручним для підтримки та подальшого розвитку.

Зважаючи на обмеження за обсягом пам’яті та потребу в оптимізації швидкості завантаження сторінок, зображення перед збереженням обробляються на сервері. Для цього використовується бібліотека sharp, яка дозволяє змінювати

розміри зображень, стискати їх без суттєвої втрати якості та, за потреби, конвертувати у сучасні формати [38]. Уся ця обробка виконується до моменту відправлення зображення до хмарного сховища, що забезпечує ефективне використання ресурсу та зручність для кінцевого користувача.

Такий підхід дозволяє мінімізувати навантаження на інфраструктуру, забезпечити швидке завантаження зображень у браузері, знизити витрати на зберігання та гарантувати гнучкість системи в майбутньому. Завдяки цьому рішення виглядає сучасним, безпечним і відповідає найкращим практикам веб-розробки.

Для забезпечення зручної та однорідної роботи над проєктом у різних середовищах використаємо контейнери Docker [5]. Це дозволить ізолювати додаток і всі його залежності від операційної системи, а також спростити процес налаштування оточення. Для керування контейнерами та організації їх взаємодії застосуємо docker-compose.

Спершу налаштуємо базовий docker-compose файл для локальної розробки. Він визначає сервіси, необхідні для роботи бекенд-серверу, зокрема базу даних PostgreSQL та кеш Redis. Конфігурація містить налаштування образів, змінних середовища, портів та томів для збереження даних. Використання окремого файлу з параметрами оточення забезпечує безпеку та гнучкість налаштувань, дозволяючи легко змінювати параметри без необхідності редагувати сам docker-compose.yml.

Базований compose файл для локальної розробки виглядає наступним чином:

```
version: ‘3’
services:
    db:
        image: postgres
        restart: always
        env_file: ./server/.env
        environment:
            POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
            POSTGRES_USER: ${POSTGRES_USER}
            POSTGRES_DB: ${POSTGRES_DB}
        ports:
            - ${POSTGRES_PORT}:5432
        volumes:
            - ./pgdata:/var/lib/postgresql/data
```

```
redis:
    container_name: redis
    image: redis
    restart: always
    ports:
        - ${REDIS_PORT}:6379
```

Наступним кроком буде створення та налаштування трьох окремих проєктів: HTTP сервера, WebSocket сервера для роботи в реальному часі та клієнтської частини. Спочатку визначимо їхню базову структуру, а потім підключимо всі необхідні залежності — потрібні бібліотеки та інструменти для стабільної роботи. Для кожного проєкту окремо буде створено файл змінних оточення, що дозволить гнучко керувати конфігурацією і забезпечить безпеку збереження важливих параметрів.

Роботу над проектом слід розпочати з реалізації функціоналу авторизації, реєстрації та інших дій, пов’язаних безпосередньо з користувачем. Для забезпечення належного рівня безпеки та зручності цього процесу було обрано механізм авторизації на основі JSON Web Token (JWT) [25]. У нашій реалізації передбачено використання двох типів токенів: access і refresh.

Access токен має короткий термін дії (15 хвилин) і використовується для авторизації при кожному запиті до захищених маршрутів API. Його зберігання відбувається у LocalStorage на клієнті, що дозволяє швидко вставляти токен у заголовки запитів.

Refresh токен має триваліший термін дії (30 днів) і зберігається у HttpOnly cookie, що захищає його від доступу з JavaScript і тим самим мінімізує ризик викрадення. Додатково refresh токен зберігається на сервері в базі даних, що дозволяє відкликати його вручну (наприклад, при виході користувача з системи або зміні пароля).

Якщо термін дії access токена спливає, на клієнті спрацьовує механізм автоматичного оновлення. За допомогою refresh токена, надісланого у cookie, клієнт непомітно для користувача надсилає запит на сервер на оновлення пари токенів. У відповідь сервер перевіряє refresh токен у базі даних, і якщо все в порядку — видає нові access та refresh токени. Таким чином користувач не помічає жодних переривань у роботі.

На клієнтській частині важливою складовою цього механізму є перехоплювачі запитів (interceptors), які автоматично додають access токен до заголовків, а також перехоплюють 401 помилку (Unauthorized). У разі помилки вони автоматично ініціюють оновлення токенів і повторюють запит уже з новим access токеном. Цей процес виглядає наступним чином:

```
const requestAuthInterceptor = (config: InternalAxiosRequestConfig) => {
  const accessToken = localStorage.getItem(ACCESS_TOKEN);
  config.headers.Authorization = `Bearer ${accessToken}`;

  return config;
};

export const authApi = axios.create({ baseURL: API_URL, withCredentials: false });

authApi.interceptors.request.use(requestAuthInterceptor);

authApi.interceptors.response.use(
  (resp) => resp,
  async (error) => {
    const originalRequest = error.config;
    if (error.response.status === 401 && !error.config._retry) {
      error.config._retry = true;
      try {
        const { accessToken } = await refreshToken();
        localStorage.setItem(ACCESS_TOKEN, accessToken);
        return authApi.request(originalRequest);
      } catch {
        console.error("Error refreshing token", error);
      }
    }
    throw error;
  }
);
```

Це забезпечує автоматичне, непомітне для користувача оновлення токенів у разі закінчення терміну дії access токена, а також повторне виконання початкового запиту з оновленими авторизаційними даними.

Схематично весь процес взаємодії з токенами, включно з оновленням доступу та обробкою помилок, зображено на відповідній діаграмі (див. Додаток В).

Додатково було реалізовано функціонал підтвердження електронної пошти, що є обов’язковим кроком для завершення реєстрації. Якщо користувач не підтвердив пошту, його обліковий запис вважається неверифікованим. У межах цього блоку також реалізовано авторизацію через Google за допомогою

бібліотеки passport.js, можливість зміни пароля, скидання забутого пароля, механізм оновлення токенів і вихід із системи з відкликанням refresh токена [39].

Паралельно з розробкою серверної частини ведеться створення клієнтського інтерфейсу, в якому інтегровані всі зазначені процеси. Завдяки грамотно спроектованій схемі авторизації користувач отримує безпечний і стабільний доступ до системи без необхідності повторного входу при кожному оновленні токена.

Після створення основних дій з користувачем, наступним кроком стає розроблення повноцінного функціоналу для роботи з оголошеннями. Цей функціонал включає створення, додавання до обраного та порівняння оголошень, їх редагування та видалення. Важливо, щоб інтерфейс для роботи з оголошеннями був спроектований максимально зручним і інтуїтивним, оскільки оголошення містять великий обсяг даних, які користувач повинен легко вводити та переглядати.

При створенні оголошення реалізуємо інтеграцію з зовнішнім API, що виконає автоматичне розпізнавання номерного знаку автомобіля, що значно спростить процес введення даних та зробить процес продажу більш прозорим [33]. Також реалізуємо інтеграцію із зовнішнім сервісом для підвантаження доступних марок автомобілів залежно від року випуску, а також моделей, які дійсно існують для конкретного року і марки [34]. Це дозволить уникнути помилок і вводити тільки актуальні дані про транспортні засоби.

Крім того, запровадимо систему модерації оголошень. Кожне оголошення проходить через наступні статуси:

- pending — оголошення очікує перевірки модератором або адміністратором; під час цього статусу редагування оголошення заборонене;
- approved — оголошення підтверджене і доступне для публікації на сайті;

- rejected — оголошення відхилене з вказівкою причини відхилення; продавець може відредагувати оголошення та повторно надіслати на перевірку.

Після створення оголошення воно потрапить у статус “pending”, що означає очікування перевірки. Адміністратори або модератори переглядають оголошення і можуть змінити його статус на “approved”, або “rejected” з зазначенням причини відхилення. У випадку відхилення продавець має можливість відредагувати оголошення відповідно до зауважень, після чого повторно надіслати його на перевірку. Важливо, що під час статусу “pending” оголошення не можна редагувати, щоб забезпечити модераторам перегляд актуальних і незмінних даних під час перевірки.

Схематично послідовність дій при створенні оголошення зображено на рис. 2.3.

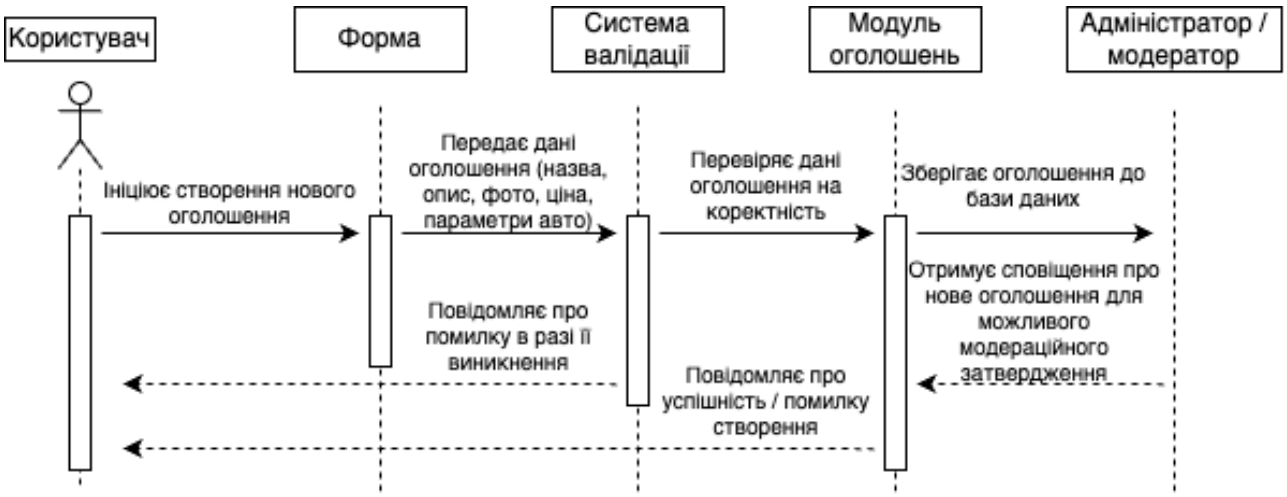


Рис. 2.3 – Діаграма послідовності для створення оголошення

Для підвищення зручності інформування користувачів, а також розширення каналів комунікації, реалізована інтеграція із офіційним Telegram-каналом сайту. Як тільки оголошення отримує статус “approved”, система автоматично надсилає до цього каналу повідомлення. Повідомлення містить основну інформацію про автомобіль та фото, що робить його доступним для перегляду користувачами без необхідності заходити на веб-сайт. Такий підхід підвищує зручність моніторингу нових оголошень, адже багато користувачів активніше користуються месенджерами, аніж веб-інтерфейсом.

Після того, як процес створення та модерації оголошень було налагоджено, розпочнемо розробку головної сторінки сайту, де відображаються всі доступні оголошення, а також сторінки з детальним переглядом конкретного оголошення. При цьому враховано, що продавець може деактивувати оголошення, і такі записи не повинні відображатися у публічному списку. Такий комплексний підхід забезпечує зручну, безпечну та контрольовану роботу з оголошеннями на сайті.

Як було зазначено раніше, наш веб-сервер повинен швидко обробляти запити та надавати користувачу необхідні дані. Для цього реалізуємо механізм посторінкової навігації, а також фільтрацію і сортування результатів. Це дозволить користувачу отримувати лише релевантні дані та зберігати параметри пошуку для зручності.

Проте, навіть із цими оптимізаціями, навантаження на сервер може залишатися значним. Тому необхідно додати проміжний рівень між клієнтом і сервером, який зможе ефективно обробляти часті або ресурсоємні запити. Цей рівень — кешування, яке буде реалізоване за допомогою Redis [24].

Принцип роботи кешу полягає в тому, що перед зверненням до бази даних ми спочатку перевіряємо наявність необхідних даних у кеші. Якщо дані знайдені — вони негайно повертаються клієнту, що значно пришвидшує відповідь. Якщо ж даних у кеші немає — виконується запит до бази, а отримані результати кешуються для подальшого швидкого доступу. Водночас кеш потребує періодичного оновлення або очищення, адже дані в системі змінюються. Наприклад, щойно видалене оголошення необхідно одразу вилучити з кешу, щоб користувачі не отримували застарілу інформацію.

З метою зменшення кількості запитів до сервера та підвищення чутливості інтерфейсу, на клієнтській стороні застосовано бібліотеку TanStack Query [19]. Вона забезпечує зручний і ефективний механізм роботи з асинхронними даними, зокрема автоматичне кешування результатів запитів. Завдяки цьому, якщо певні дані вже були отримані, повторні звернення до сервера протягом певного часу не відбуваються, що знижує навантаження на бекенд і пришвидшує відображення

інформації для користувача. Нижче наведено приклад конфігурації глобальних параметрів клієнта TanStack Query, у якому визначено, що кешовані дані залишаються актуальними протягом 10 хвилин:

```
export const queryClient = new QueryClient({
  defaultOptions: {
    queries: { staleTime: 10 * 60 * 1000 },
  },
});
```

Крім цього, при виконанні змін даних (наприклад, створення, редагування або видалення оголошень) реалізовано механізм так званих “оптимістичних оновлень”, який дозволяє миттєво оновлювати інтерфейс, не чекаючи завершення відповіді від сервера. Суть підходу полягає в тому, що до виконання запиту на зміну даних клієнтська частина застосунку зберігає поточний стан кешу (наприклад, список оголошень або окрему сутність), а потім одразу оновлює ці дані у кеші відповідно до очікуваного результату. Детальний приклад реалізації оптимістичного оновлення кешу під час видалення оголошення наведено в додатку Г.

Якщо сервер успішно обробляє запит — зміни зберігаються, і додатково відбувається оновлення відповідних запитів для синхронізації кешу з актуальним станом на сервері. У випадку помилки — в обробнику помилок виконується відкат кешованих даних до попереднього збереженого стану. Такий підхід забезпечує більш плавну взаємодію з інтерфейсом та зменшує відчутну затримку для користувача, зберігаючи водночас цілісність і достовірність даних у системі.

Так як ми вже маємо можливість створювати оголошення, тепер можна реалізувати функціонал додавання відгуків про продавців та їх перегляд. Кожен відгук буде містити заголовок, текстову частину та оцінку продавця за шкалою від 1 до 5, що дозволить формувати загальний середній рейтинг кожного продавця. Також слід передбачити можливість видалення відгуків.

Для забезпечення безпеки та запобігання неправомірному контенту, адміністратори та модератори мають право видаляти відгуки. Такий контроль необхідний для підтримки якості інформації на платформі та захисту користувачів від недоброчесного чи неприйнятного контенту.

Також слід реалізувати функціонал редагування профілю користувача, який включатиме можливість оновлення особистих даних, зміни пароля та відновлення пароля у випадку його втрати. При цьому зміна електронної пошти не передбачена, оскільки вона є основним контактним каналом та другим основним ідентифікатором користувача у багатьох внутрішніх сервісах і процесах. Такий підхід гарантує стабільність і цілісність даних користувача, а також запобігає потенційним проблемам із прив'язкою інформації до облікового запису.

Після завершення основної роботи з HTTP-сервером можна перейти до розробки функціоналу чату. Чат забезпечить зручний спосіб комунікації між клієнтом та продавцем автомобіля в режимі реального часу.

Для уникнення необхідності постійного оновлення сторінки браузера для перевірки нових повідомлень, використовується протокол WebSocket [20]. Він дозволяє встановити постійне з'єднання між клієнтом і сервером, що дає змогу надсилати та отримувати повідомлення миттєво. З метою спрощення роботи з WebSocket реалізацією застосовується бібліотека socket.io, яка інкапсулює складність протоколу та надає зручний API як для серверної, так і для клієнтської частини.

Для оптимізації роботи сервера передбачено механізм посторінкового завантаження повідомлень: замість отримання всіх повідомлень одразу користувачеві спочатку завантажуються останні 20. При досягненні верхньої межі чату з'являється кнопка для завантаження попередніх 20 повідомлень і так далі, доки не буде завантажено весь чат. Такий підхід дозволить значно знизити навантаження на сервер і покращити продуктивність клієнтського інтерфейсу.

Для адміністраторів слід передбачити окремий інтерфейс — адміністративну панель, яка дозволить ефективно керувати основними елементами системи. Реалізація цього функціоналу включає дві основні вкладки: "Оголошення" та "Користувачі". У розділі оголошень адміністратор матиме доступ до повного переліку всіх створених оголошень, зокрема тих, що перебувають у статусах pending, rejected, та деактивованому стані, які не

відображаються на головній публічній сторінці. Для кожного оголошення передбачено можливість перегляду основної інформації, переходу до детальної сторінки самого оголошення або профілю продавця, а також виконання дій — активація, деактивація чи видалення.

У вкладці користувачів буде реалізовано перегляд списку всіх зареєстрованих користувачів з ключовими відомостями про кожного. Для кожного користувача можна буде відкрити детальну сторінку, де адміністратор зможе змінити його роль, заблокувати або розблокувати обліковий запис. Блокування користувачів із роллю адміністратора або модератора буде обмежено системою, щоб уникнути порушення стабільної роботи панелі керування. Додатково на цій сторінці відображатиметься статистика щодо активності користувача: кількість поточних оголошень (включно з деактивованими та тими, що перебувають в статусах rejected та pending), кількість видалених записів (тих, що користувач створював протягом всього часу користування платформою).

Слід зазначити, що кожна з описаних вище операцій — як на клієнтській, так і на серверній стороні — супроводжується обробкою помилок, перевірками та валідацією вхідних даних: від хешування паролю, до перевірки розміру та формату зображень. Всі дані зберігаються у базі даних із дотриманням вимог надійності та безпеки. У ряді випадків доступ до певних операцій (наприклад, перегляд користувачів, всіх оголошень) обмежено і надається лише окремим групам користувачів відповідно до їх ролей у системі. Подібні обмеження реалізовані і на клієнтській частині: окремі сторінки або функції доступні лише авторизованим користувачам з відповідними правами доступу.

Усі функціональні можливості були реалізовані з урахуванням сучасних технологій, стандартів безпеки, принципів масштабованості та кращих практик розробки веб-застосунків, що забезпечує надійну та зручну роботу системи в цілому.

Висновки до другого розділу

У другому розділі було проведено детальний аналіз функціональних і нефункціональних вимог до веб-платформи для розміщення оголошень про продаж автомобілів. Розглянуто особливості взаємодії з системою відповідно до ролей користувачів. Архітектура платформи побудована з урахуванням зручності використання, надійного управління даними та можливості масштабування, що є ключовими аспектами для сучасного онлайн-сервісу.

У межах розділу визначено вимоги до продуктивності, зокрема час реакції системи при реєстрації, пошуку та створенні оголошень, що гарантує стабільну роботу платформи в умовах навантаження. Значну увагу приділено гнучкості архітектури, яка дозволяє розширювати функціонал без негативного впливу на швидкодію.

Для візуалізації логіки роботи було використано діаграми активностей, які відображають ключові бізнес-процеси, зокрема створення та перевірку оголошень, їх фільтрацію та взаємодію адміністратора з контентом. Діаграма класів дозволила проілюструвати структуру системи та взаємозв’язки між її основними модулями.

Було описано структуру реляційної бази даних PostgreSQL з чітко визначеними таблицями, полями та зв’язками. Розглянуто питання безпечної авторизації користувачів за допомогою двох JWT-токенів, що забезпечує захищений контроль сесій.

Особливу увагу приділено оптимізації взаємодії з даними: реалізовано посторінкове завантаження для зниження навантаження на сервер, додано фільтрацію для швидкого доступу до потрібної інформації, впроваджено серверне кешування за допомогою Redis, а також клієнтське кешування з використанням TanStack Query, включаючи підтримку оптимістичних оновлень.

Таким чином, результати другого розділу формують стійке технічне та концептуальне підґрунтя для реалізації платформи, що відповідає актуальним вимогам до зручності, продуктивності, масштабованості та безпеки.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ІЗ СЕРВІСОМ

3.1. Порядок встановлення та налаштування параметрів системи

Для забезпечення стабільної та коректної роботи веб-платформи з розміщення оголошень про продаж автомобілів необхідно провести належне розгортання всіх її компонентів та налаштування взаємодії між ними. Архітектура платформи реалізована із застосуванням технології Docker, що дозволяє кожному компоненту функціонувати у власному ізольованому середовищі. Це значно спрощує процеси оновлення, масштабування, забезпечує легке розгортання системи в різних середовищах, а також полегшує її подальшу підтримку. Для візуалізації архітектури та взаємодії між основними елементами створено діаграму розгортання, представлену на рис. 3.1.

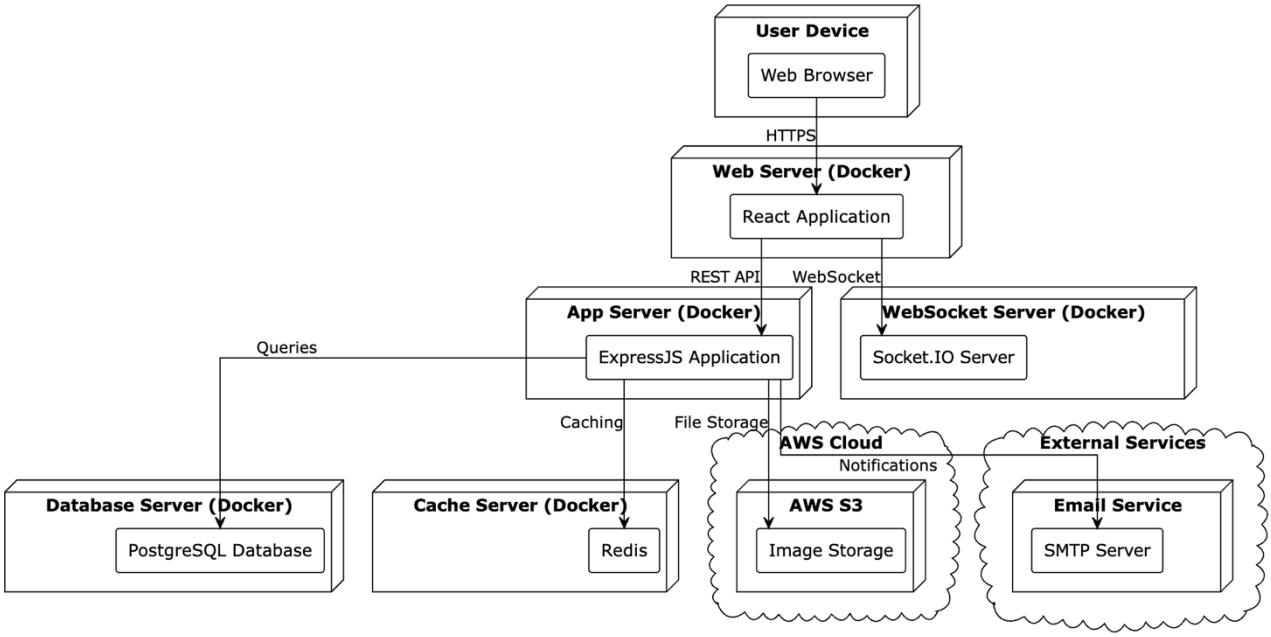


Рис. 3.1 – Діаграма розгортання веб-платформи для розміщення оголошень про продаж автомобілів

Користувачі взаємодіють із платформою за допомогою веб-браузера, що є основним інструментом доступу до клієнтського інтерфейсу. Цей інтерфейс реалізовано за допомогою React-застосунку, який розгорнутий у Docker-контейнері та виконується на веб-сервері. Через нього здійснюється обробка запитів до API, а також підтримка з’єднання з WebSocket-сервером для обміну даними в реальному часі.

Запити, що надходять від клієнта, передаються до бекенд-серверу, на якому розгорнутий застосунок на основі ExpressJS. Цей застосунок також ізольовано у Docker-контейнері та відповідає за всю бізнес-логіку системи, обробку HTTP-запитів, авторизацію, взаємодію з базою даних, кешем та зовнішніми сервісами. Окремий WebSocket-сервер, який функціонує у власному контейнері, відповідає за обмін даними в реальному часі, включаючи підтримку чату та відображення статусу онлайн користувачів. React-застосунок напряму встановлює WebSocket-з'єднання з цим сервером, минаючи основний API.

Для зберігання структурованих даних використовується база даних PostgreSQL, яка функціонує у власному контейнері. Бекенд-застосунок напряму звертається до цієї бази, виконуючи необхідні запити для обробки даних оголошень, користувачів та ін. З метою оптимізації продуктивності та зменшення навантаження на базу даних у системі також використовується Redis — система кешування, яка дозволяє зберігати часто використовувані запити та інші тимчасові дані, що потрібні для функціонування системи.

Для зберігання медіафайлів, зокрема фотографій автомобілів, які додають користувачі, застосовується хмарне сховище AWS S3. Зв'язок із ним здійснюється з боку бекенду, який передає зображення до сховища та зберігає відповідні назви файлів в базі даних.

Крім того, платформа інтегрується з зовнішнім поштовим сервісом (SMTP), який відповідає за надсилання системних електронних листів: підтверджень реєстрації, сповіщень про відновлення пароля [23].

Уся система організована як сукупність взаємопов'язаних сервісів, кожен із яких виконує окрему роль і функціонує у власному контейнері. Це забезпечує модульність, легкість у масштабуванні та підтримці, а також дозволяє незалежно оновлювати окремі компоненти без ризику порушення роботи всієї платформи. Такий підхід відповідає сучасним вимогам до веб-архітектур і гарантує надійність та продуктивність при роботі з великим навантаженням.

3.2. Структура інтерфейсу веб-застосунку

Інтерфейс є важливою складовою будь-якого проєкту, оскільки він визначає зручність користування та взаємодію користувачів із платформою. У цьому розділі представлено структуру інтерфейсу нашого сервісу, яка базується на сучасних принципах дизайну та UI/UX практиках.

Структура інтерфейсу охоплює навігаційні елементи, візуальне оформлення, керування контентом та інші функціональні компоненти, що формують користувацький досвід. Далі детальніше розглянемо ці аспекти.

Після входу на сайт нас зустрічає головна сторінка, на якій відображаються всі оголошення, які є на сайті (рис. 3.2).

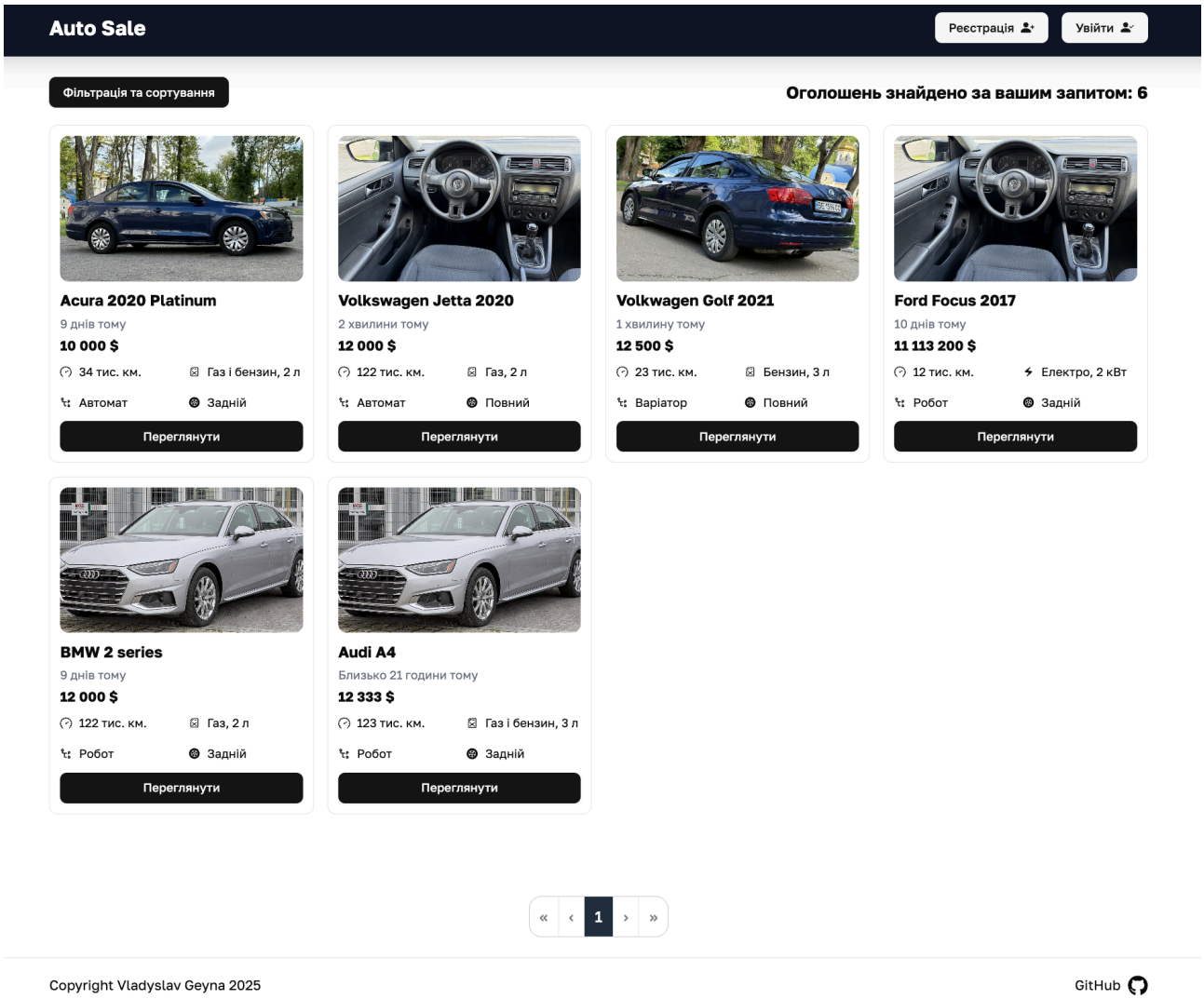


Рис. 3.2 – Головна сторінка сайту

Над ними є кнопка “Фільтрація та сортування”, натиснувши на яку, ми побачимо вспливаюче вікно, де можна обрати потрібні нам параметри фільтрації та / або сортування або скинути їх до стану по замовчуванню (рис. 3.3).

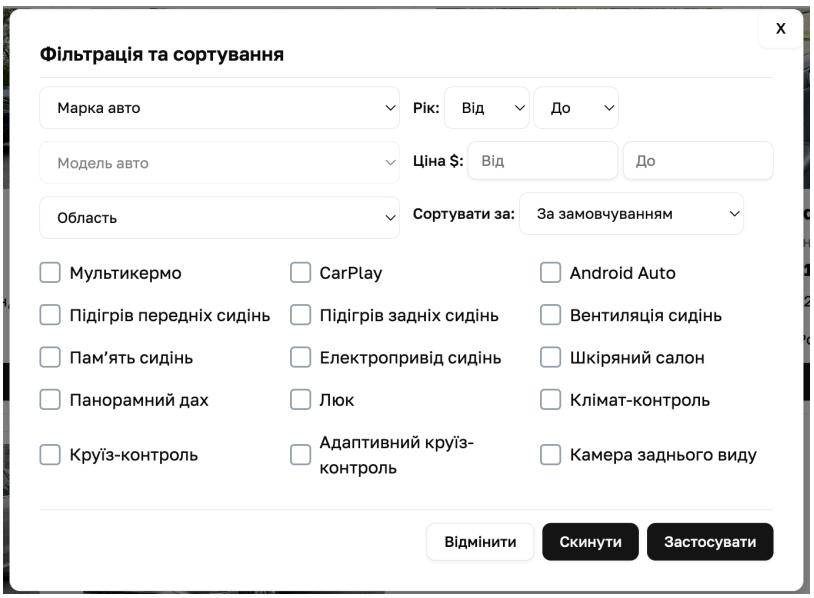


Рис. 3.3 – Фільтрація та сортування

Для того, щоб ми мали можливість повноцінно користуватись сайтом зареєструємось. Для цього перейдемо на сторінку реєстрації натиснувши на відповідну кнопку в шапці сайту. В разі помилки введення після відправки форми ми побачимо відповідне повідомлення над нею (рис. 3.4). Якщо помилку можна відслідкувати на етапі введення, то вона буде відображена безпосередньо під полем вводу.

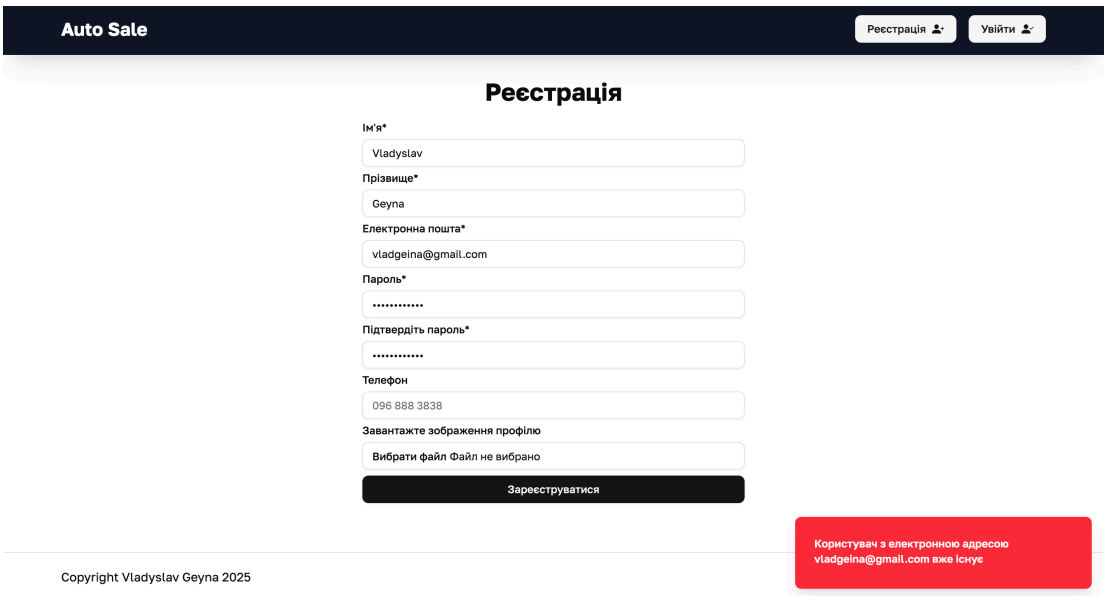


Рис. 3.4 – Сторінка реєстрації з прикладом помилки

Спробуємо зареєструватись. Після успішної реєстрації нам на вказану електронну адресу приходить лист із посиланням для підтвердження облікового запису. Після переходу по посиланню нас переадресує на сторінку авторизації (рис. 3.5). У випадку із Google реєстрацією / авторизацією нас одразу переадресує на головну сторінку сайту без потреби підтвердження email.

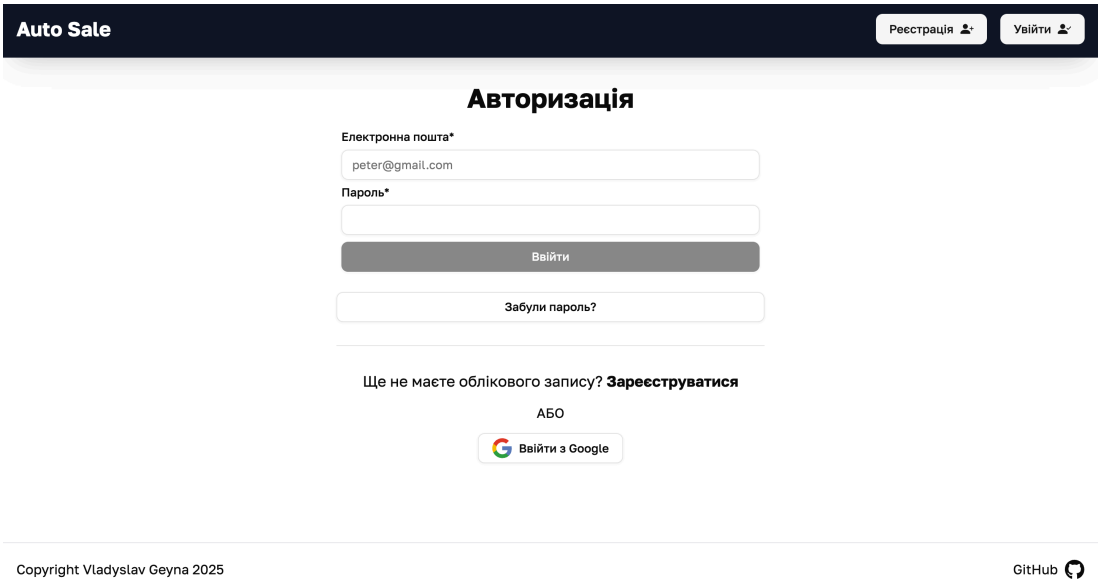


Рис. 3.5 – Сторінка авторизації

Тепер спробуємо ввійти. Навіть якщо першочергово ми зареєструвались не за допомогою Google, ми все ще можемо ввійти з його допомогою при наявності цього email в списку доступних.

Якщо з тих, чи інших причин пароль до облікового запису було втрачено ми можемо, натиснувши на кнопку “Забули пароль?”, відновити його, надіславши лист на вказану електронну адресу (рис. 3.6).

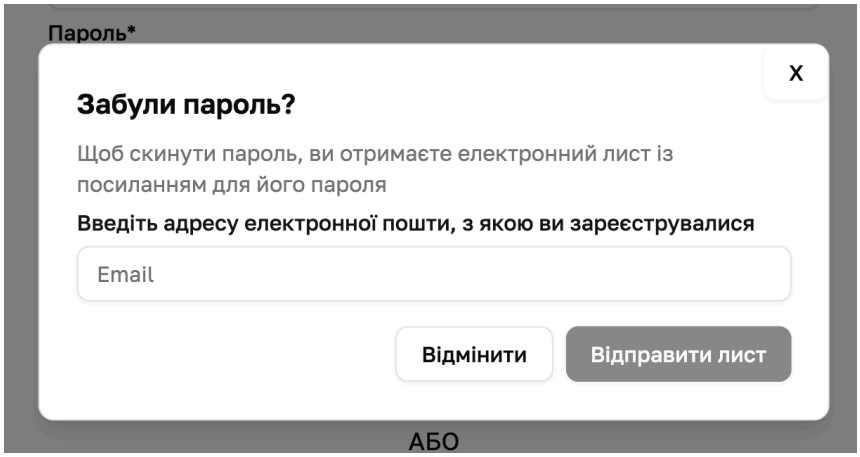


Рис. 3.6 – Форма для відправки листа для скидання паролю

Після успішного входу нас зустрічає та ж головна сторінка, проте із іншими елементами в шапці сайту (рис. 3.7).

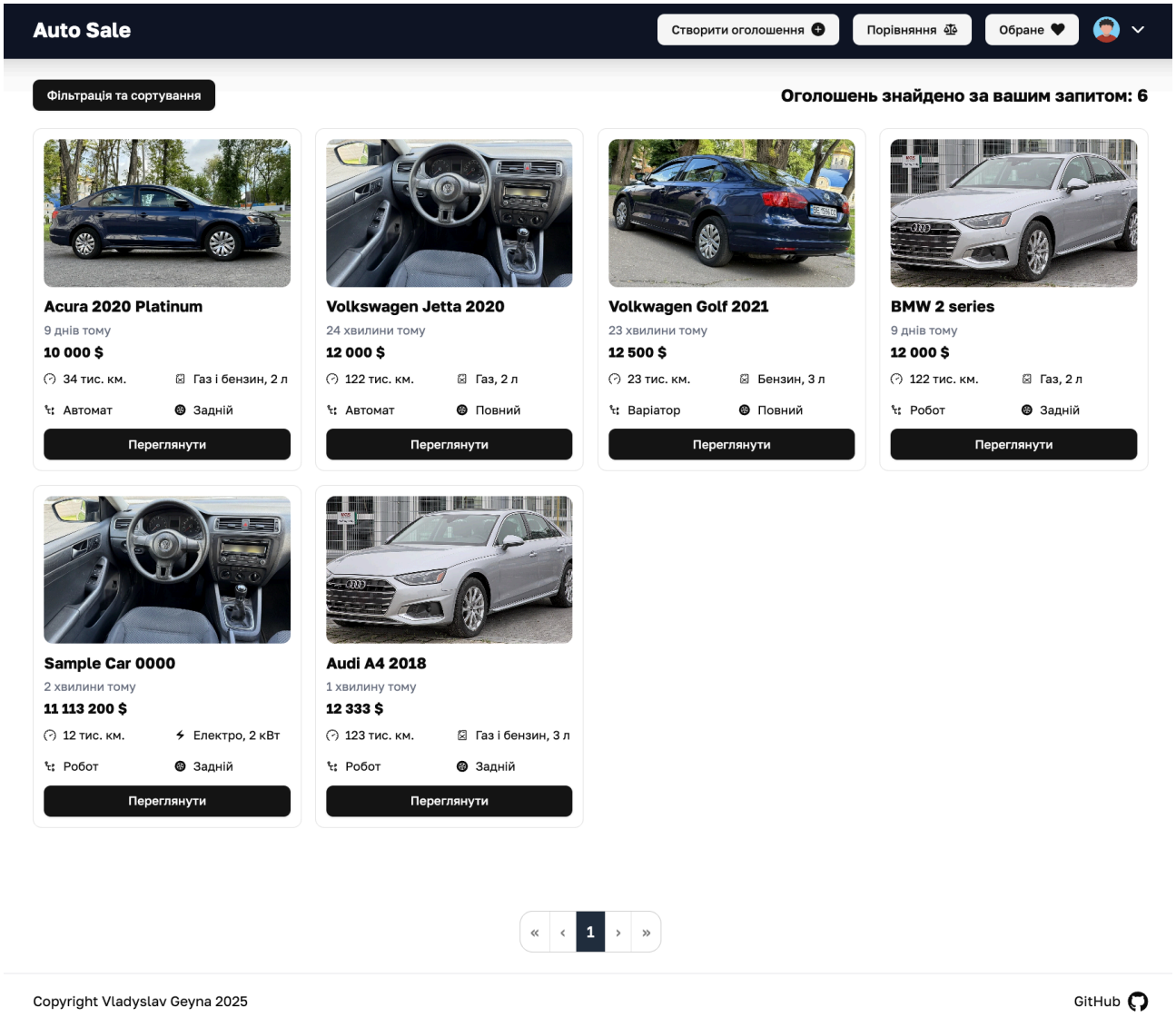


Рис. 3.7 – Головна сторінка сайту після входу

В випадаючому меню, яке знаходиться в шапці сайту, містяться посилання на наступні сторінки (рис. 3.8):

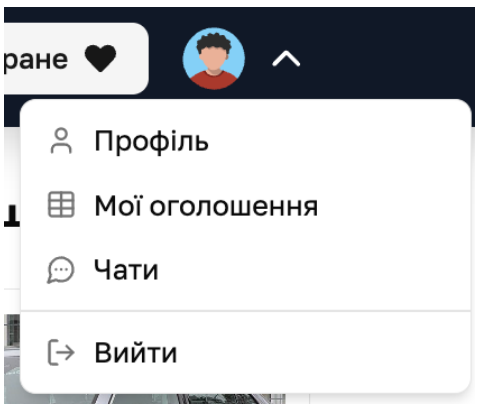


Рис. 3.8 – Випадаюче меню

Розглянемо їх пізніше. Зараз акцентуємо увагу на функціоналі створення оголошення. Натиснемо на кнопку в шапці “Створити оголошення”. Бачимо наступну сторінку:

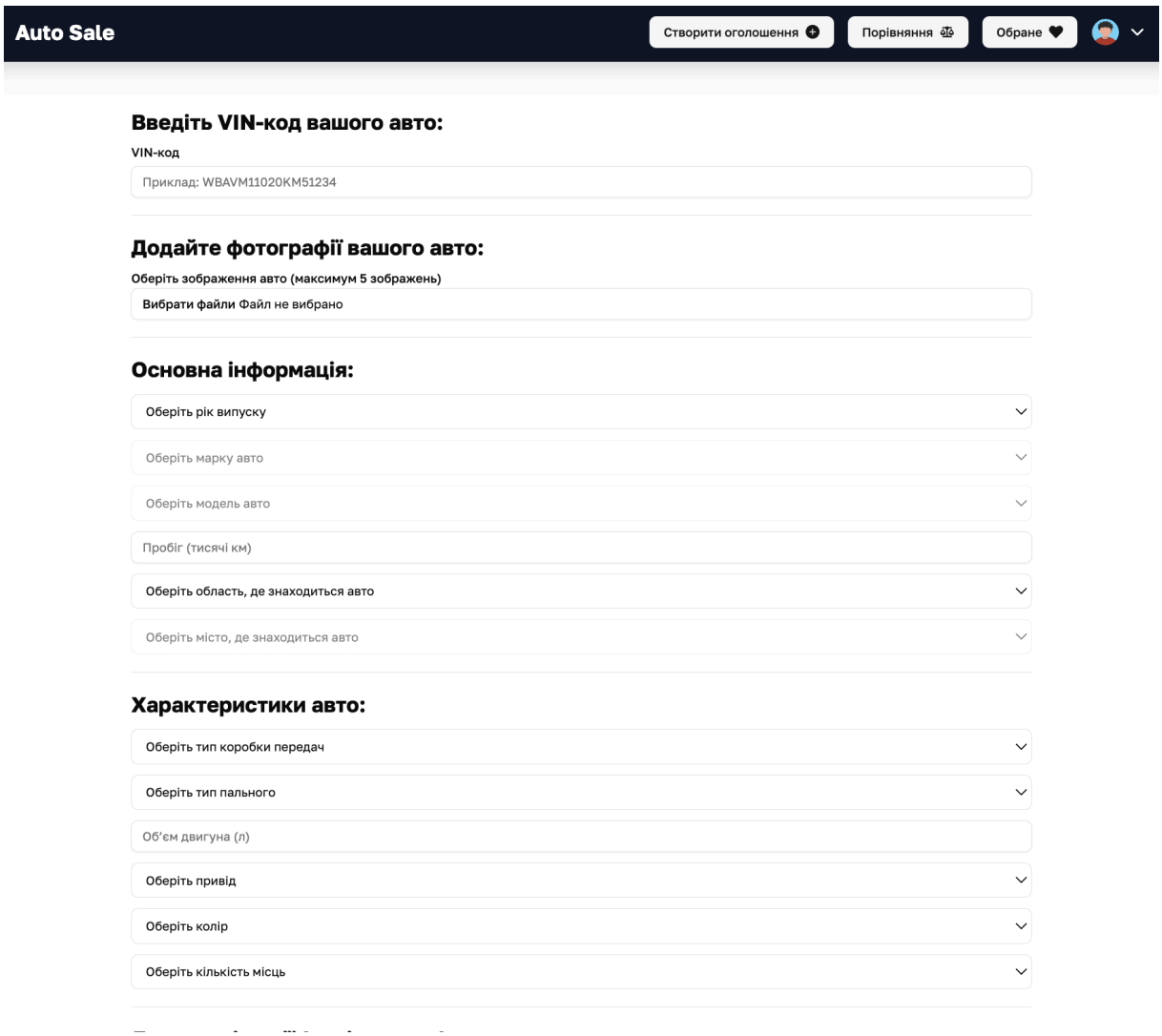


Рис. 3.9 – Сторінка додавання оголошення

Додамо декілька оголошень. Цей процес було спрощено, додаванням підтримки автоматичного розпізнавання номерного знаку авто по фото та підвантаженням доступних марок авто за роком та моделей – за маркою та роком. Після успішного додавання оголошення ми будемо повідомлені про це відповідним вспливаючим блоком зеленого кольору з текстом.

Тепер у випадаючому меню перейдемо до розділу “Мої оголошення”. Тут відображаються додані нами записи (рис. 3.10). Варто зазначити, що після додавання оголошення воно переходить на етап модерації адміністратором або модератором.

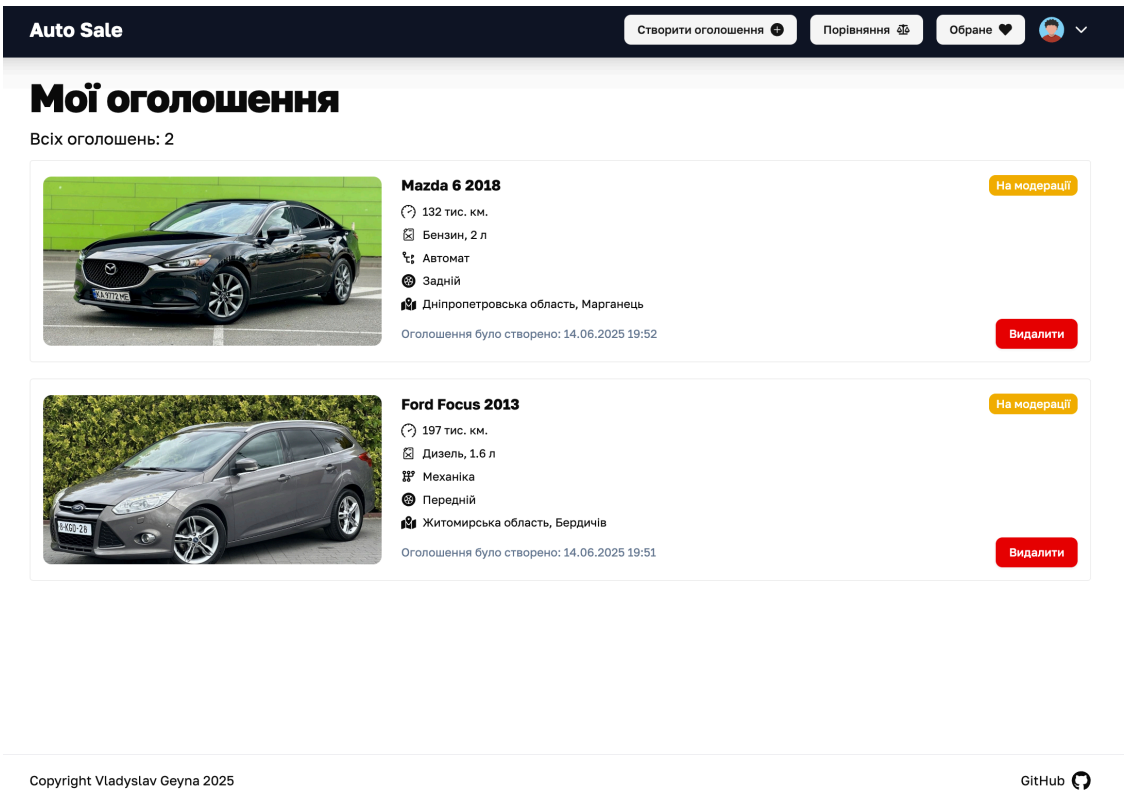


Рис. 3.10 – Перегляд моїх оголошень

У цей період редагування, активація або деактивація оголошення недоступні, залишається лише можливість його видалення. При спробі виконати цю операцію (а також інші безповоротні чи ризиковані дії в подальшому на сайті) користувач побачить спливаюче вікно підтвердження, як показано на рис. 3.11.

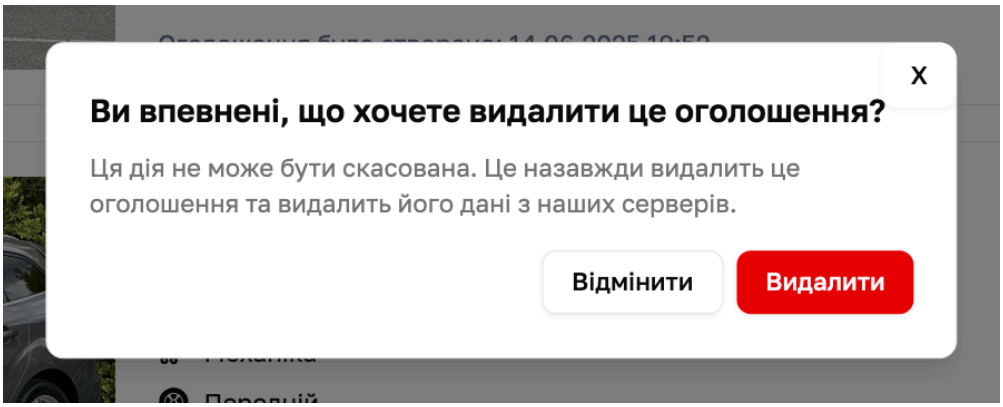


Рис. 3.11 – Підтвердження видалення

Перейдемо до сторінки оголошення. Якщо воно належить нам, його відображення виглядає так, як показано на рис. 3.12.

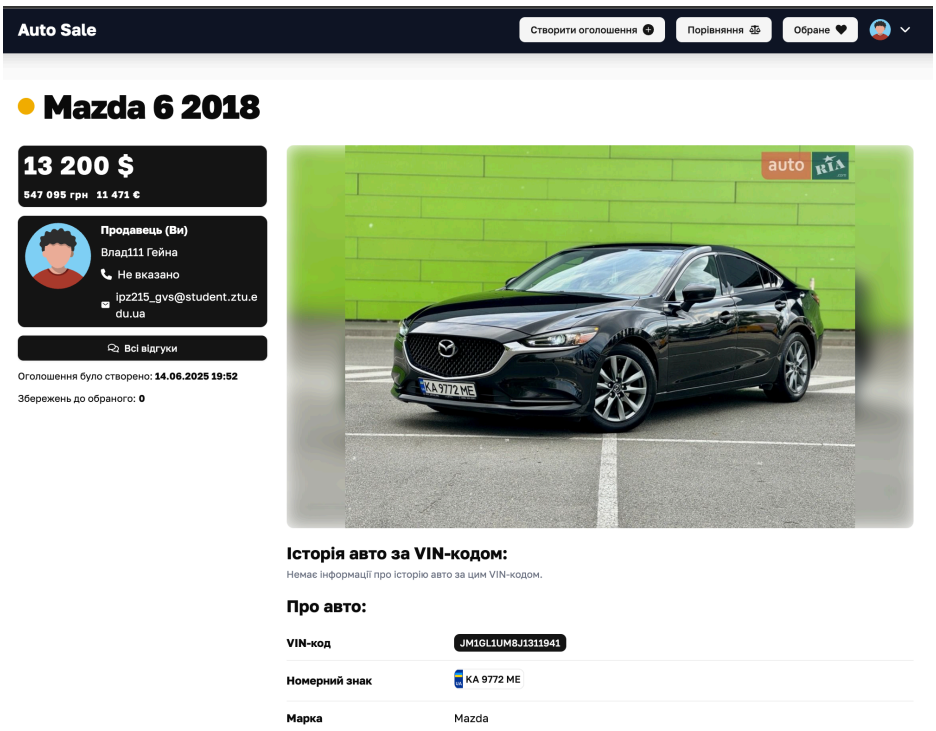


Рис. 3.12 – Перегляд доданого нами оголешення

Якщо ж це оголошення іншого користувача — воно відображається інакше, як на рис. 3.13.

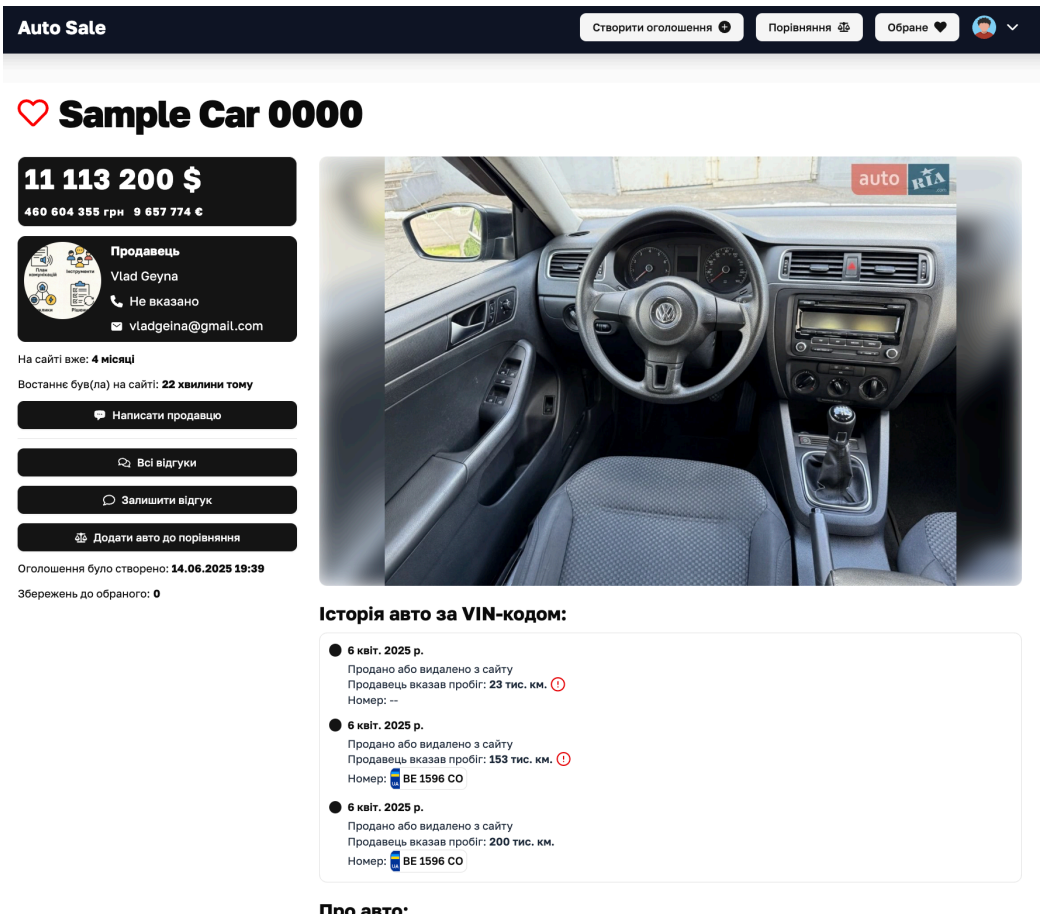


Рис. 3.13 – Перегляд оголешення доданого не нами

Тут можна переглянути коротку інформацію та контакти продавця, його поточний онлайн-статус або час останнього перебування в мережі, а також загальний час перебування на сайті. Окрім цього, відображається інформація про автомобіль: його ціна, зображення, опис та, зокрема, номерний знак, який був ідентифікований автоматично та інша інформація. Важливо зазначити, що система зберігає історію авто за VIN-кодом, тож якщо певне авто виставляється на продаж повторно, ми маємо можливість ознайомитися з його історією, зокрема з показниками пробігу. Зліва також розміщена інформація про дату створення оголошення та кількість його додавань до обраного.

Над цими даними розташовані чотири кнопки (для власного оголошення доступна лише одна). Перша дозволяє розпочати або продовжити чат із продавцем — цей функціонал буде детальніше описаний далі в межах цього розділу. Друга кнопка відкриває всі відгуки про продавця, третя дає змогу залишити відгук, а четверта — додати або видалити авто з порівняння. Біля заголовку оголошення знаходиться піктограма у вигляді серця. Натиснувши на неї, можна додати авто до обраного — піктограма заповниться суцільним кольором. Повторне натискання видалить оголошення з обраного, і колір піктограма повернеться до початкового.

Почнемо з відгуків. Натиснемо на кнопку “Залишити відгук” та побачимо сторінку (рис. 3.14).

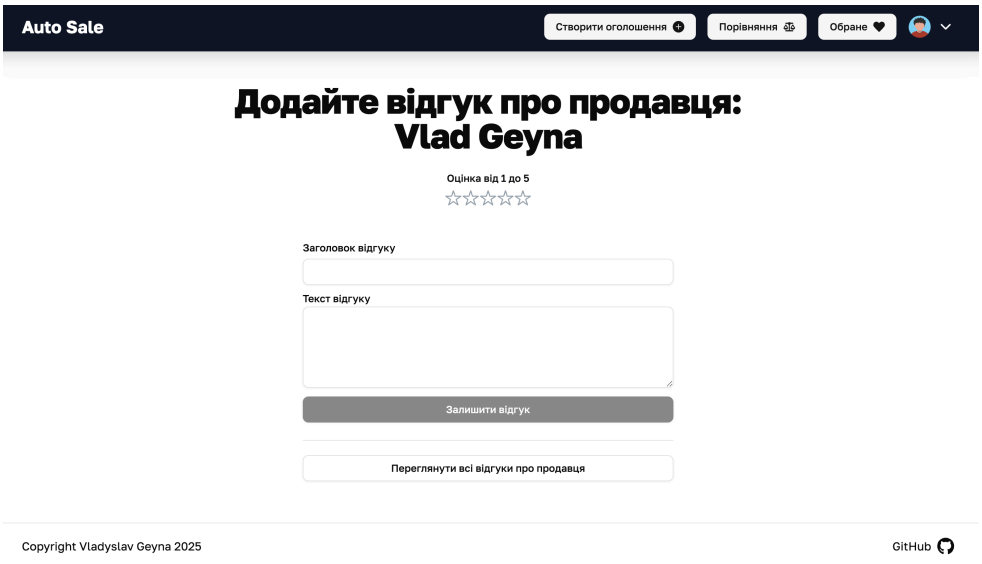


Рис. 3.14 – Залишення відгуку про продавця

Спробуємо залишити відгук, після чого нас буде перенаправлено на сторінку з усіма відгуками про цього продавця (рис. 3.15). Це еквівалентно переходу за посиланням “Всі відгуки” на сторінці перегляду оголошення. Варто зазначити, що для коректного формування рейтингу можна залишити лише один відгук на продавця.

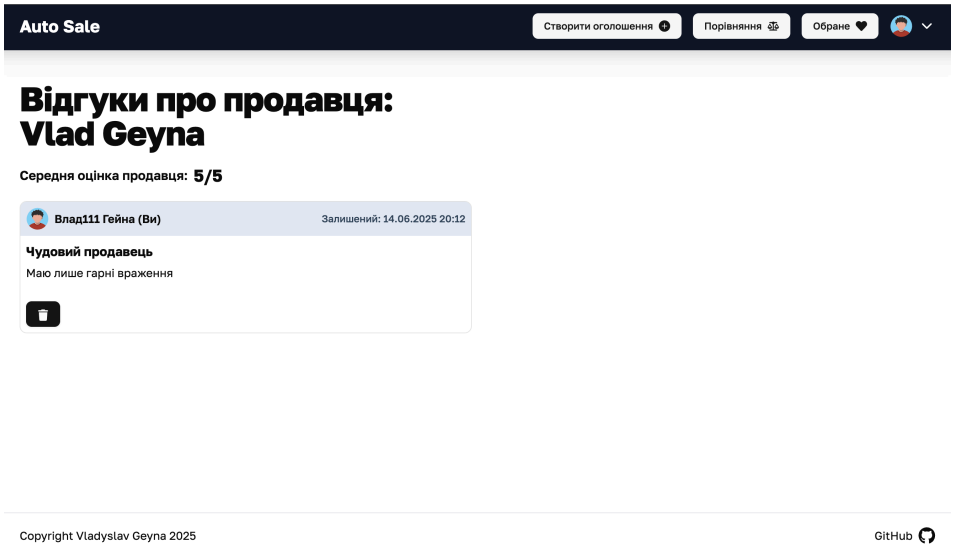


Рис. 3.15 – Перегляд відгуків про продавця

Відгуки, залишені нами, ми можемо видаляти. Інші ж лише переглядати.

Тепер спробуємо додати декілька авто до обраного та порівняння, натиснувши відповідно на кнопки “Додати авто до порівняння” та на піктограму серця біля заголовку оголошення. Переглянемо тепер наше обране. Для цього в шапці перейдемо по посиланню “Обране”. Бачимо список оголошень (рис. 3.16).

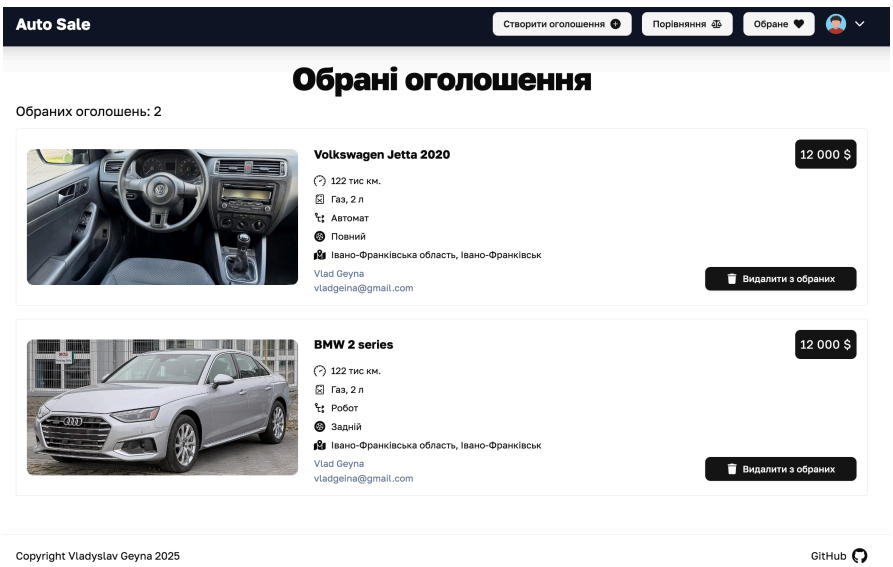


Рис. 3.16 – Сторінка з обраним

Тут при потребі ми можемо видалити оголошення з обраного.

Тепер перейдемо до сторінки із порівнянням. Для цього натиснемо на кнопку “Порівняння”, що знаходиться в шапці сайту та побачимо наступну сторінку (рис. 3.17):

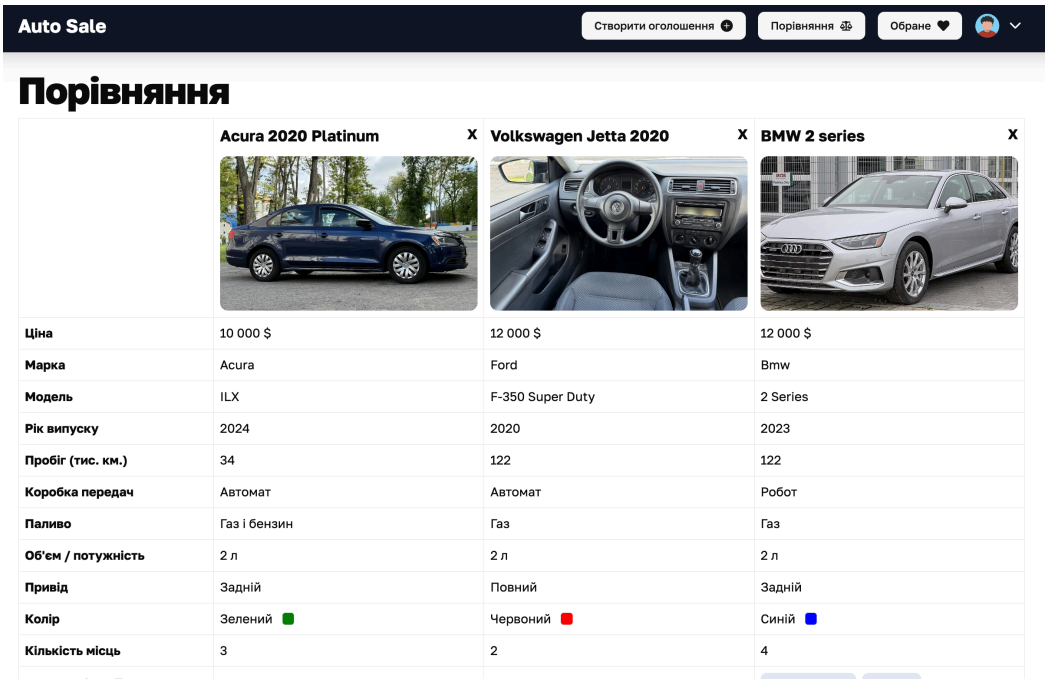


Рис. 3.17 – Сторінка з порівнянням

Тут для того, щоб видалити авто з порівняння, потрібно натиснути на піктограму в вигляді літери “X” навпроти заголовку оголошення.

Розглянемо профіль користувача. Для цього в випадаючому меню натиснемо на кнопку “Профіль”. Побачимо наступну сторінку (рис. 3.18):

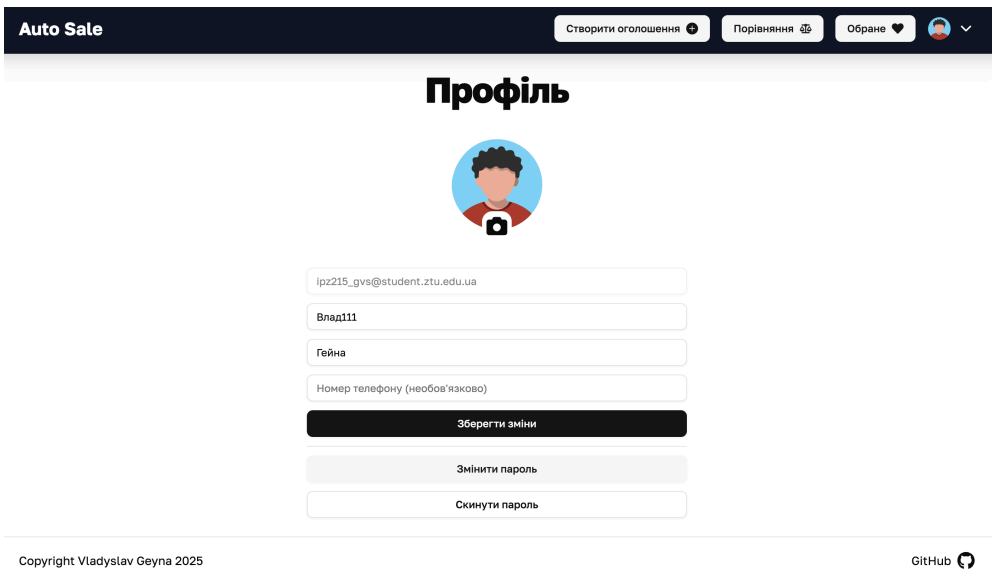


Рис. 3.18 – Профіль користувача

Опинившись на сторінці, ми бачимо автоматино підставлені поточні дані користувача. Тут ми можемо обрати або змінити аватар, ім’я, прізвище, телефон, а також змінити, чи скинути пароль. Слід зазначити, чи електронну адресу змінити не можна.

При переході за посиланням “Змінити пароль” відкривається сторінка, на якій спочатку потрібно ввести поточний пароль, а потім двічі — новий (рис. 3.19).

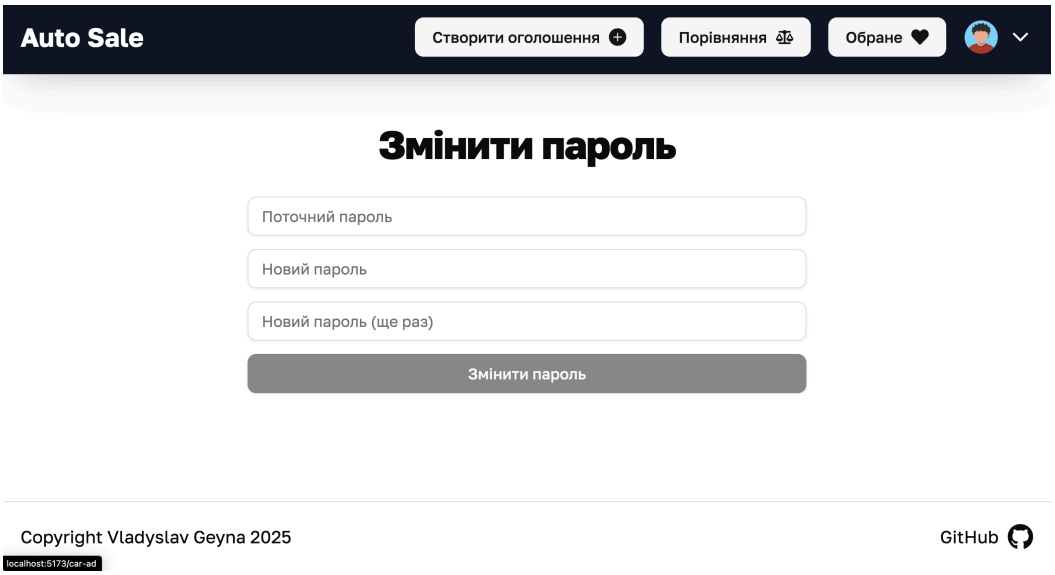


Рис. 3.19 – Зміна паролю

Якщо ж ми спробуємо скинути пароль, то нам буде запропоновано отримати лист із згенерованим посиланням для подальшого скидання паролю (рис. 3.20).

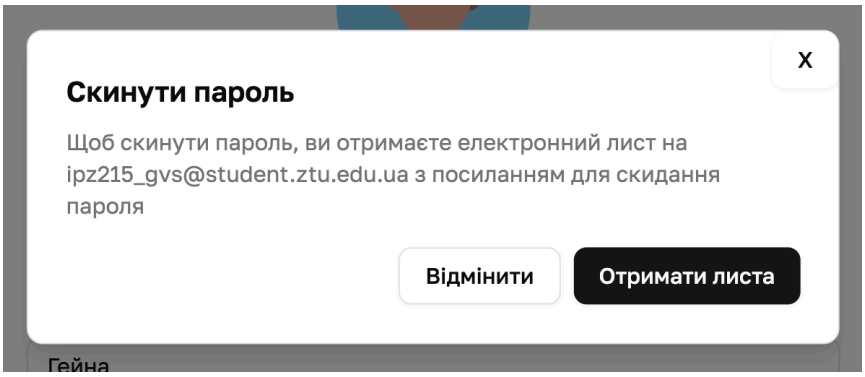


Рис. 3.20 – Підтвердження відправки листа для скидання паролю

Якщо ми погодимось, то нам прийде лист із згенерованим посиланням для скидання паролю, яке буде дійсним протягом 15 хвилин. Після переходу по посиланню ми можемо ввести новий пароль (рис. 3.21).

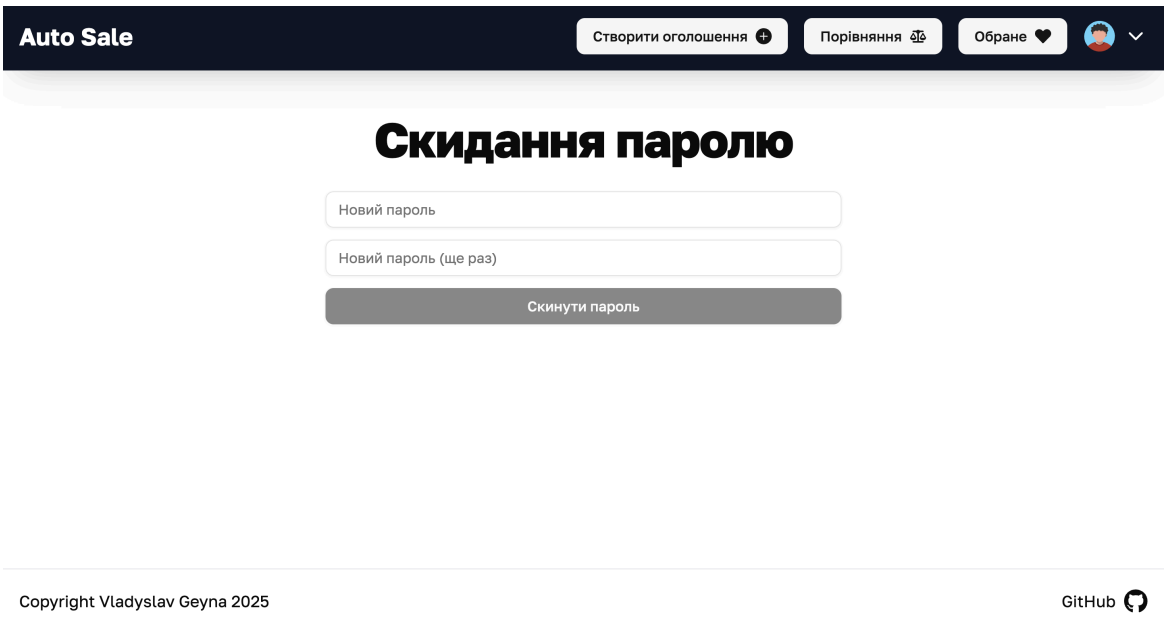


Рис. 3.21 – Скидання паролю

Розглянемо роботу чату. Для того, щоб розпочати переписку із продавцем потрібно натиснути на кнопку “Написати продавцю” на сторінці із оголошенням. Після цього нас буде переадресовано на сторінку із чатом (рис. 3.22).

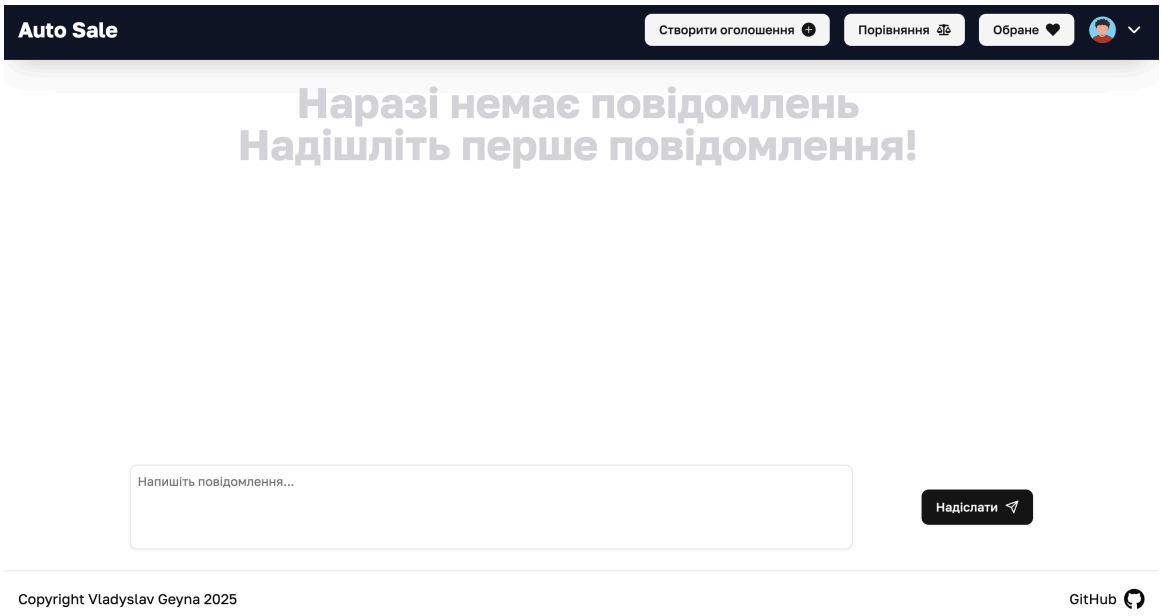


Рис. 3.22 – Вигляд чату після його створення

Якщо ж переписка із продавцем вже відбувалась раніше або чат було вже створено, то його можна буде знайти натиснувши на “Чати” в випадаючому меню. Тут ми побачимо всі існуючі чати (рис. 3.23).

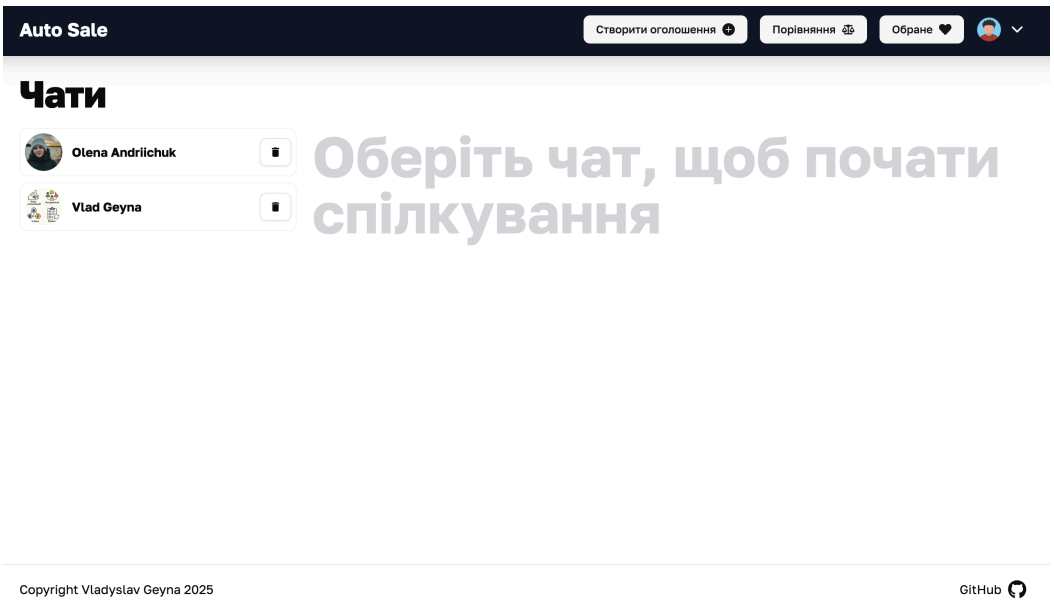


Рис. 3.23 – Список всі чатів

Спробуємо зайти в один із чатів, де відбувалась переписка і переглянути як він виглядає (рис. 3.24).

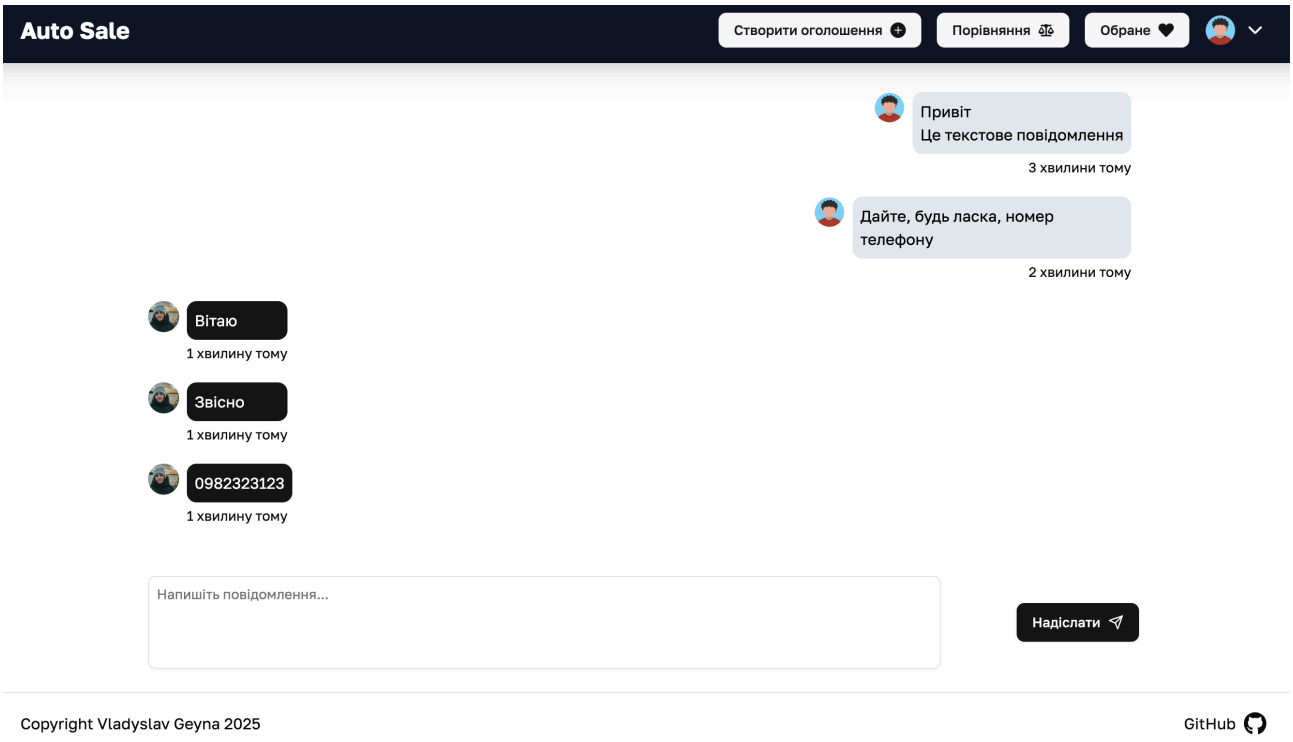


Рис. 3.24 – Сторінка чату

На сторінці із перепискою реалізовано підвантаження нових повідомлень при потребі. Тобто всього спочатку ми бачимо останні 20 повідомлень і при натисненні на кнопку зверху чату (рис. 3.25) ми отримаємо наступні 20.

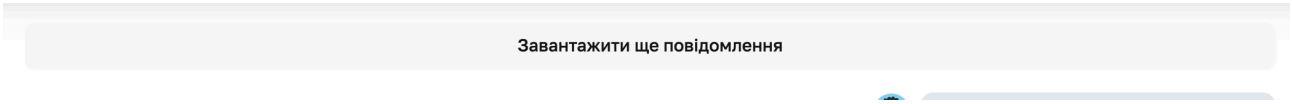


Рис. 3.25 – Завантажити повідомлення

І так доки не будуть отримані всі повідомлення (рис. 3.26).

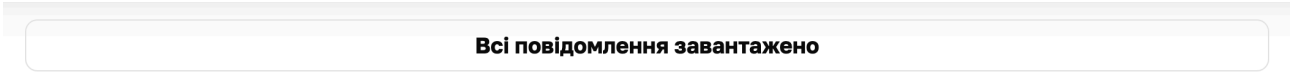


Рис. 3.26 – Завантажено всі повідомлення

На сторінці зі всіма чатами у разі наявності нових повідомлень відображається відповідний індикатор у вигляді кружечка (рис. 3.27). Тут також передбачена можливість видалити переписку. Після натискання на відповідну кнопку з’явиться спливаюче вікно для підтвердження видалення. Варто враховувати, що при підтвердженні чат буде видалено для обох учасників — як для нас, так і для співрозмовника.



Рис. 3.27 – Чат з новим повідомленням

Крім того, коли співрозмовник набирає повідомлення, це можна побачити завдяки відповідному сповіщенню (рис. 3.28).

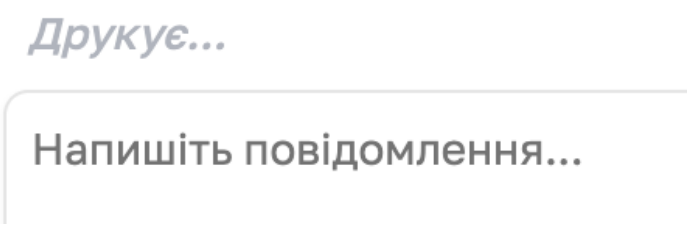


Рис. 3.28 – Сповіщення про набір повідомлення співрозмовником

Загалом можливості адміністратора та модератора на сайті такі ж, як і у звичайного користувача, однак існують деякі додаткові функції. Розглянемо їх детальніше.

Після входу для адміністратора в головному меню з’являються два додаткові пункти: “Адмін панель” та “Потребує модерації” (рис 3.29).

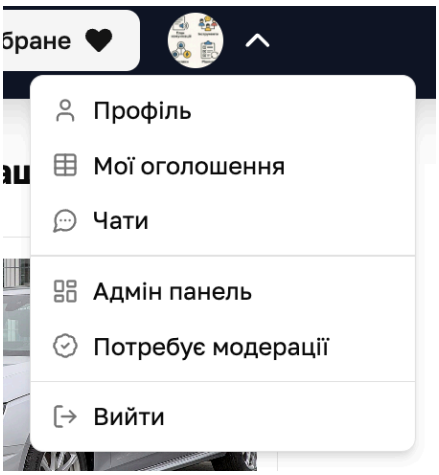


Рис. 3.29 – Випадаюче меню в адміністратора

Що стосується модератора, то з нових опцій йому доступна лише “Потребує модерації”.

Як показано на рис. 3.10, після додавання оголошення воно переходить на етап модерації. Розглянемо цей процес детальніше. Щоб розпочати модерацію нових оголошень, необхідно у випадаючому меню обрати пункт “Потребує модерації” Після цього відкриється список оголошень, які очікують на перевірку (рис. 3.30).

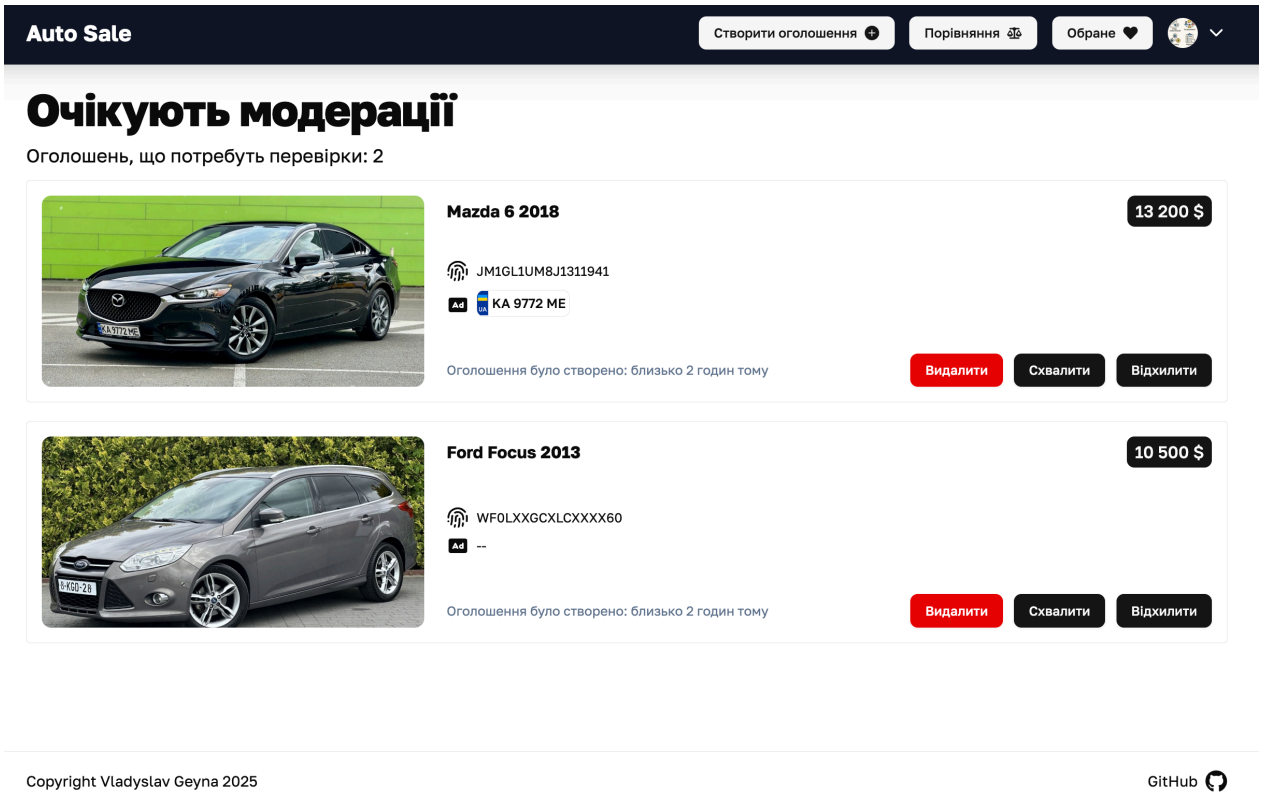


Рис. 3.30 – Оголошення, що потребують модерації

Щоб здійснити попередній перегляд оголошення, користувач може перейти на відповідну сторінку, як показано на рис. 3.12. Адміністратор або модератор має можливість одразу видалити оголошення, якщо його вміст є неприйнятним, не відповідає правилам платформи або містить зайву інформацію. Якщо під час перегляду виникають запитання чи виявлено недоліки, оголошення можна відхилити, надіславши його на доопрацювання. Для цього слід натиснути кнопку “Відхилити” та вказати причину такого рішення (рис. 3.31).

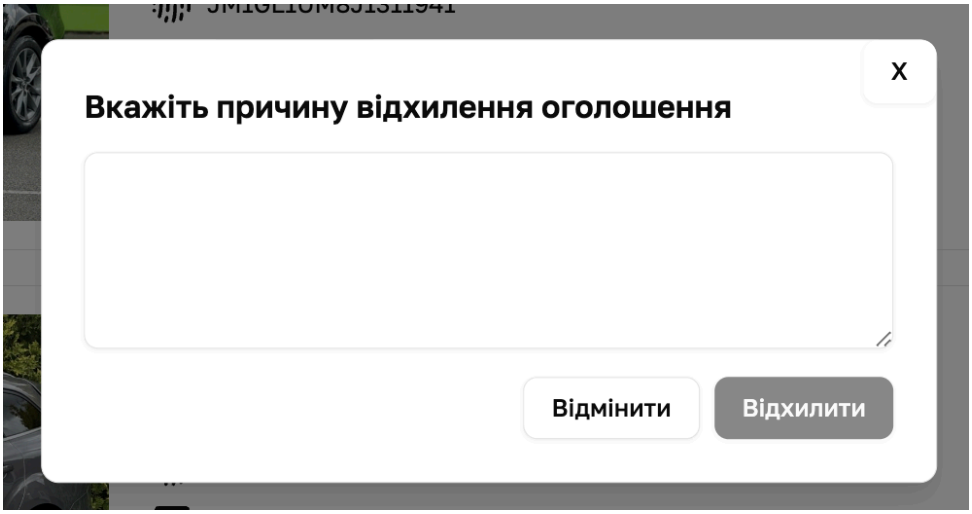


Рис. 3.31 – Введення причини відхилення оголошення

Розглянемо, як виглядає результат такої дії з боку власника оголошення. Перейшовши до розділу “Мої оголошення”, користувач бачить, що статус змінено на “Відхилено”. При наведенні курсору на цей статус відображається причина відхилення (рис. 3.32).

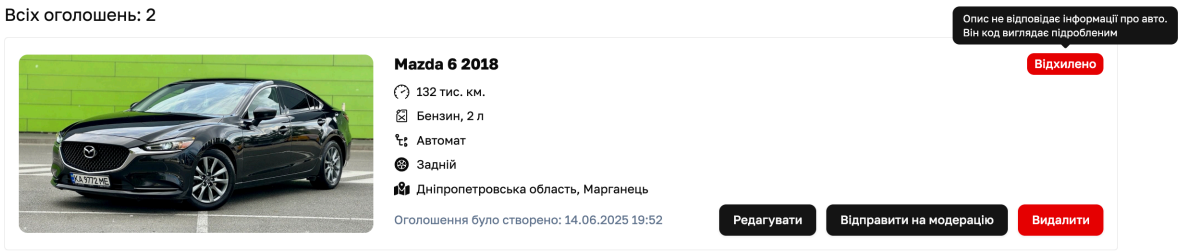


Рис. 3.32 – Відхилене оголошення

Варто зазначити, що після відхилення з’являється можливість редагування оголошення. Сторінка редагування аналогічна сторінці створення, з тією відмінністю, що всі раніше введені дані підставляються автоматично. Після

внесення необхідних змін користувач може натиснути кнопку “Відправити на модерацію” — після цього оголошення знову буде передано на перевірку відповідному користувачу.

У разі, якщо оголошення пройшло всі перевірки та не потребує жодних правок, адміністратор або модератор може натиснути кнопку “Схвалити”, після чого оголошення отримує статус “Схвалено” (рис. 3.33). Це означає, що воно публікується в загальному доступі та стає видимим для всіх користувачів.

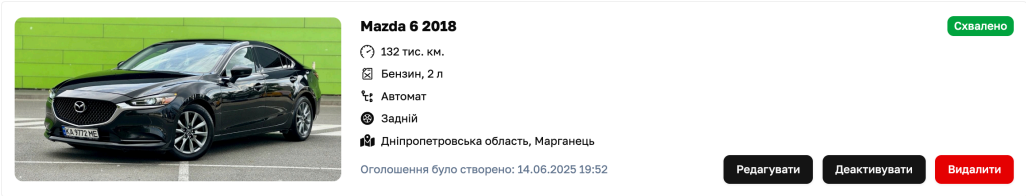


Рис. 3.33 – Схвалене оголошення

Як бачимо, після цього власник оголошення отримує можливість деактивувати його. У такому випадку воно стане недоступним для перегляду всіма користувачами, окрім самого власника, адміністратора та модератора. Функція редагування оголошення також залишається доступною, однак з метою збереження довіри до продавців і уникнення змін ключових даних, редагувати можна лише такі поля: зображення, пробіг, область і місто, заголовок, опис та ціну (рис. 3.34).

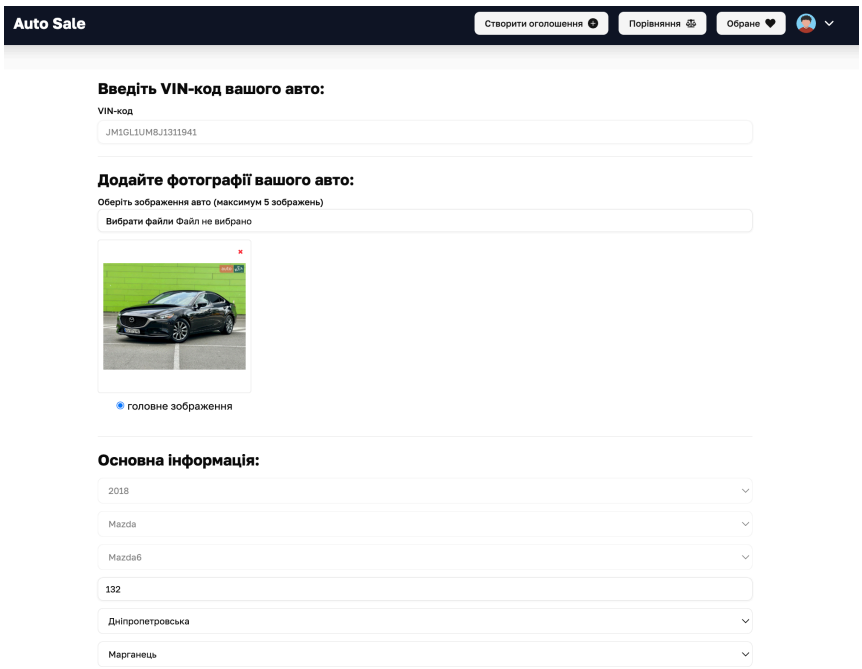


Рис. 3.34 – Редагування оголошення в статусі “Схвалено”

Сайт для розміщення оголошень про продаж автомобілів має власний Telegram-канал, що слугує додатковим каналом комунікації та підвищує зручність інформування користувачів. Зокрема, щойно оголошення отримує статус «Схвалено», система автоматично надсилає відповідне повідомлення до цього каналу (рис. 3.35). Повідомлення містить основну інформацію про автомобіль і фото, що дозволяє переглянути його без необхідності переходу на веб-сайт. Це суттєво спрощує моніторинг нових оголошень, адже багато користувачів віддають перевагу месенджерам, а не веб-інтерфейсам.

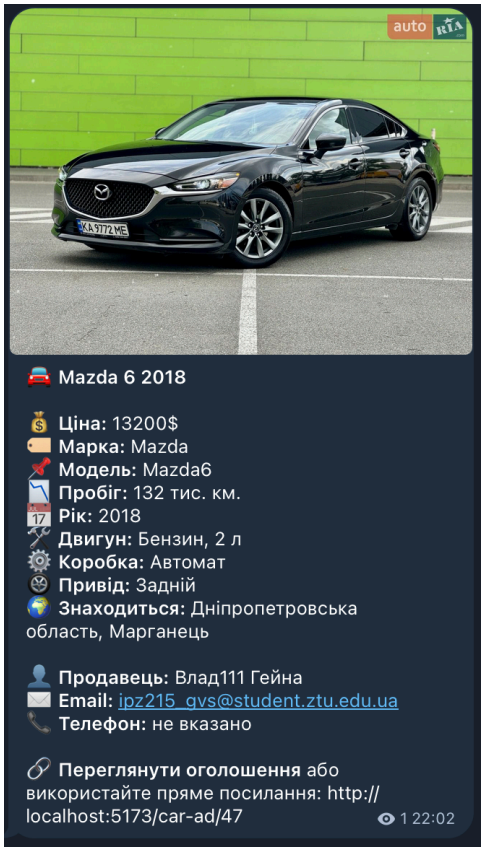


Рис. 3.35 – Оголошення, надіслане до Telegram-каналу

Тут користувач може ознайомитися з основною інформацією про авто, переглянути контакти продавця та скористатися прямим посиланням на оголошення, розміщене на веб-сайті.

Розглянемо адміністративну панель, доступ до якої адміністратор отримує через пункт «Адмін панель» у випадаючому меню.

Панель містить дві вкладки: “Користувачі” та “Оголошення”. Розглянемо функціональні можливості першої.

Список користувачів, відображений у цій вкладці, містить основну інформацію про кожного з них: ім’я, прізвище, електронну адресу, номер телефону, роль, дату реєстрації, останній візит, а також статус підтвердження облікового запису (рис 3.36).

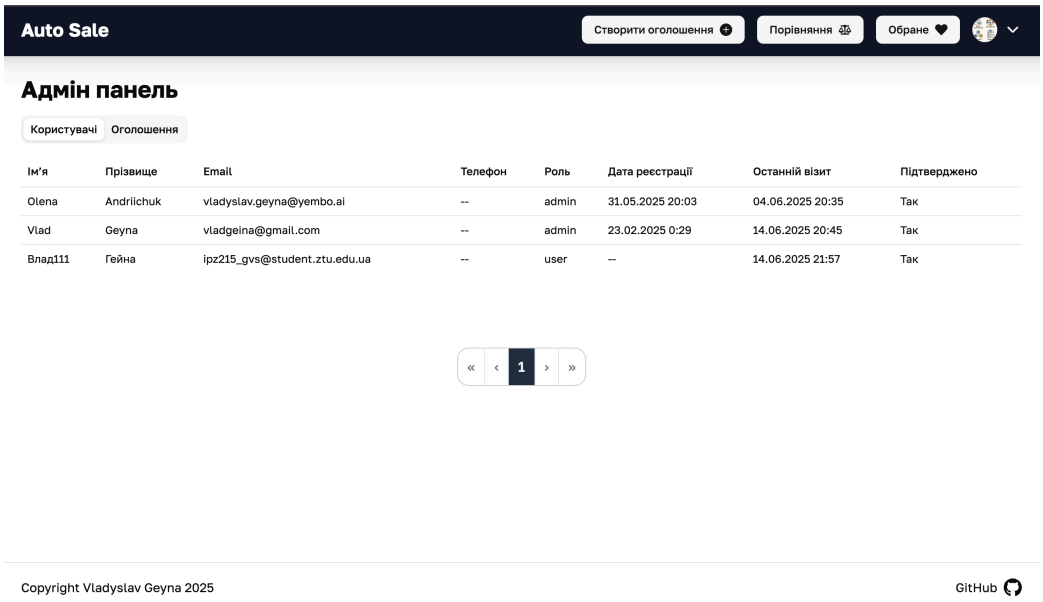


Рис. 3.36 – Адмін панель, вкладка “Користувачі”

Натиснувши на один із елементів списку, адміністратор переходить до детального перегляду інформації про конкретного користувача (рис. 3.37).

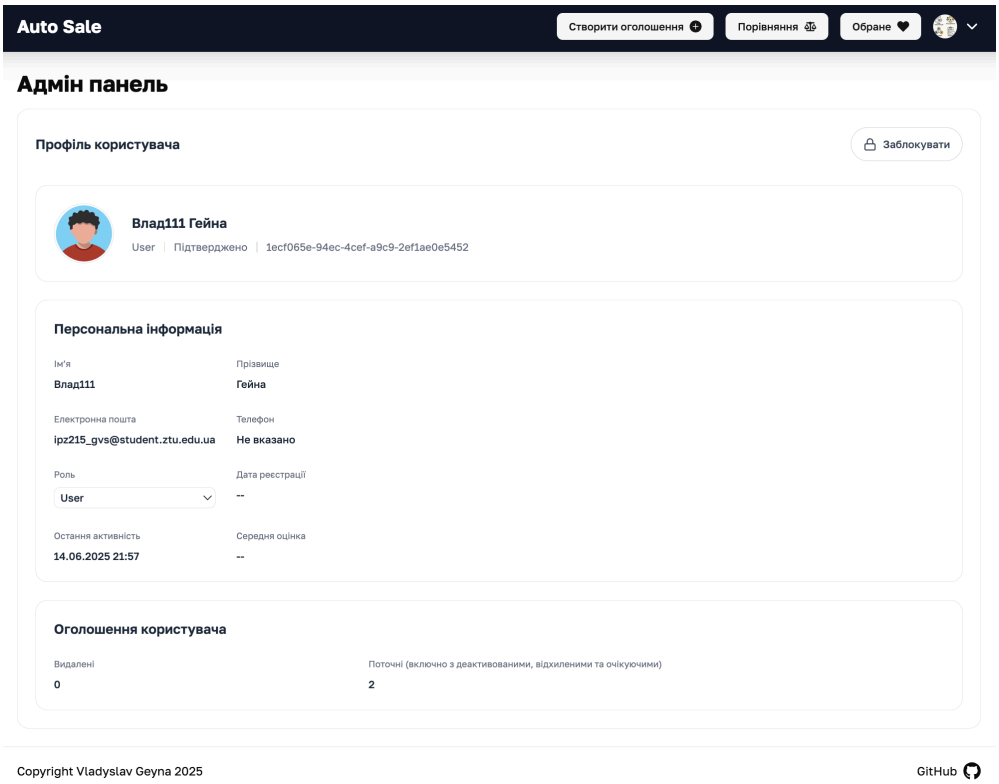


Рис. 3.37 – Огляд інформації про користувача адміністратором

На цій сторінці адміністратор може змінити роль користувача, заблокувати або розблокувати його обліковий запис. Блокування обліковок користувачів з роллю адміністратора або модератора заборонено на рівні системи — для уникнення порушення стабільної роботи панелі керування.

Окрім цього, на сторінці відображається статистика активності: кількість поточних оголошень (включно з деактивованими, відхиленими та тими, що перебувають на модерації), а також загальна кількість видалених оголошень, які користувач створив за весь час користування платформою.

Вкладка “Оголошення” надає адміністратору доступ до повного переліку всіх створених оголошень, включно з тими, що перебувають у статусах “На модерації”, “Відхилено” або знаходяться в деактивованому стані, тобто не відображаються на публічній головній сторінці.

Список оголошень у цій вкладці містить основну інформацію про кожне з них: ідентифікатор, статус, дату створення, заголовок, марку, модель, VIN-код, номер, дані продавця, а також кнопки для виконання дій — активації, деактивації або видалення (рис. 3.38)

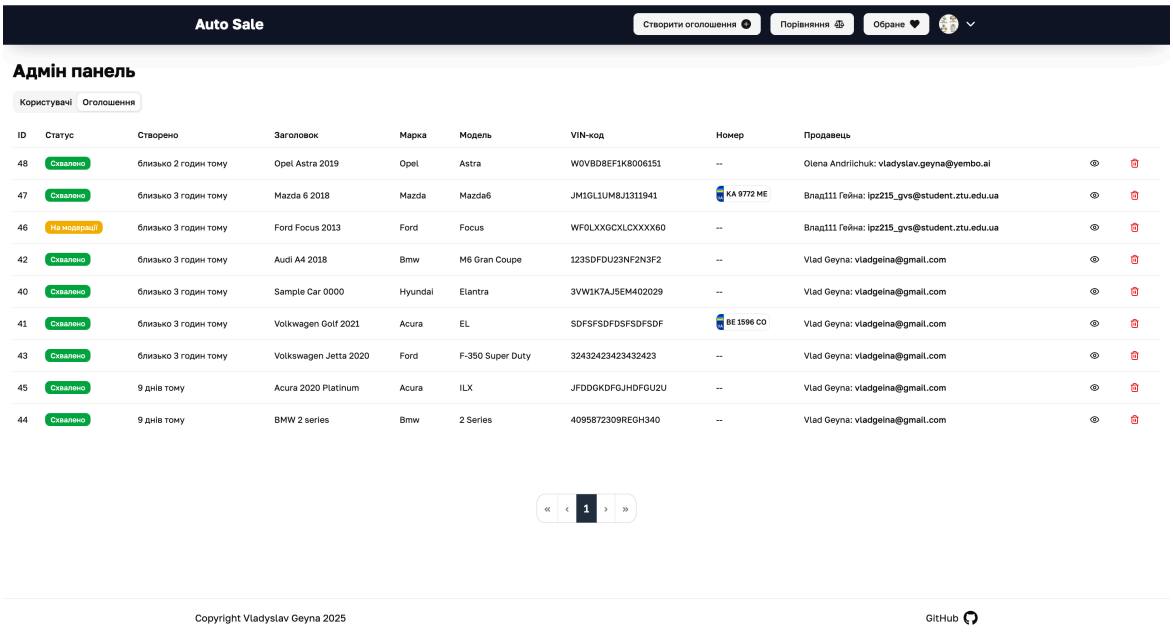


Рис. 3.38 – Адмін панель, вкладка “Оголошення”

Натиснувши на контакт продавця у відповідному рядку, можна перейти до його профілю, а натискання на сам рядок оголошення відкриває його сторінку для детального перегляду.

На завершення, щоб вийти з облікового запису, у випадаючому меню потрібно обрати пункт “Вийти”. Після натискання користувача буде переадресовано на головну сторінку сайту.

3.3. Тестування роботи програмного забезпечення

Розглянемо процес тестування ключових аспектів нашої платформи, який охоплює валідацію даних під час виконання різних операцій, авторизацію користувачів для здійснення певних запитів, а також перевірку доступу до окремих сторінок.

Валідація даних — один із критично важливих етапів тестування будь-якої платформи або застосунку [28]. Цей процес спрямований на перевірку коректності введених користувачем даних перед їх обробкою та збереженням у системі. Належна валідація дає змогу уникнути помилок, а також забезпечує узгодженість і надійність роботи платформи.

Далі розглянемо типові помилки, яких може припуститися користувач під час взаємодії з платформою.

У разі реєстрації, якщо обліковий запис із вказаною електронною адресою вже існує, або навпаки — під час авторизації, якщо вказані дані не відповідають жодному користувачу, система виведе відповідне повідомлення (рис. 3.39, рис. 3.40).

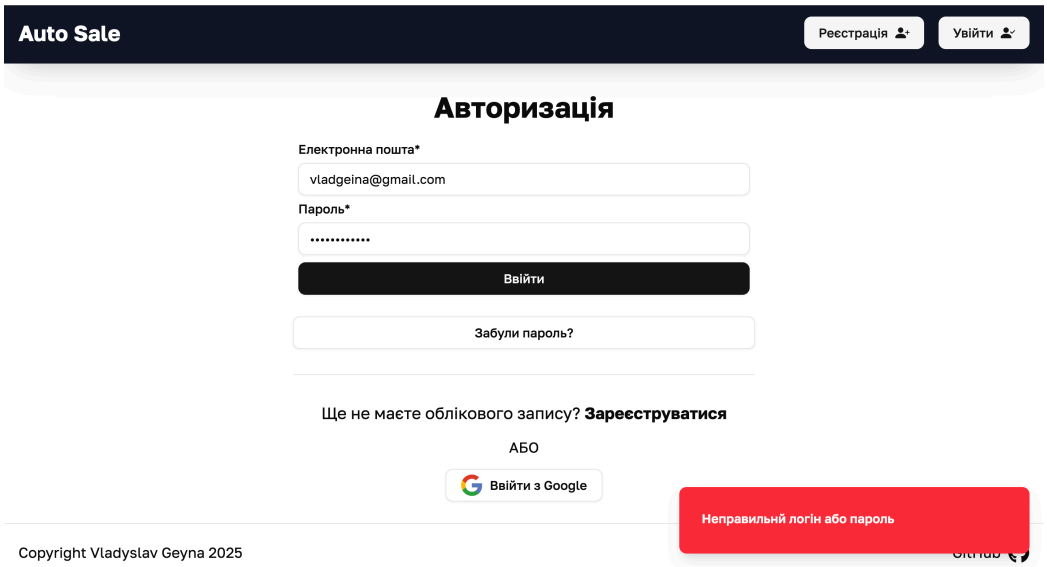


Рис. 3.39 – Помилка при авторизації

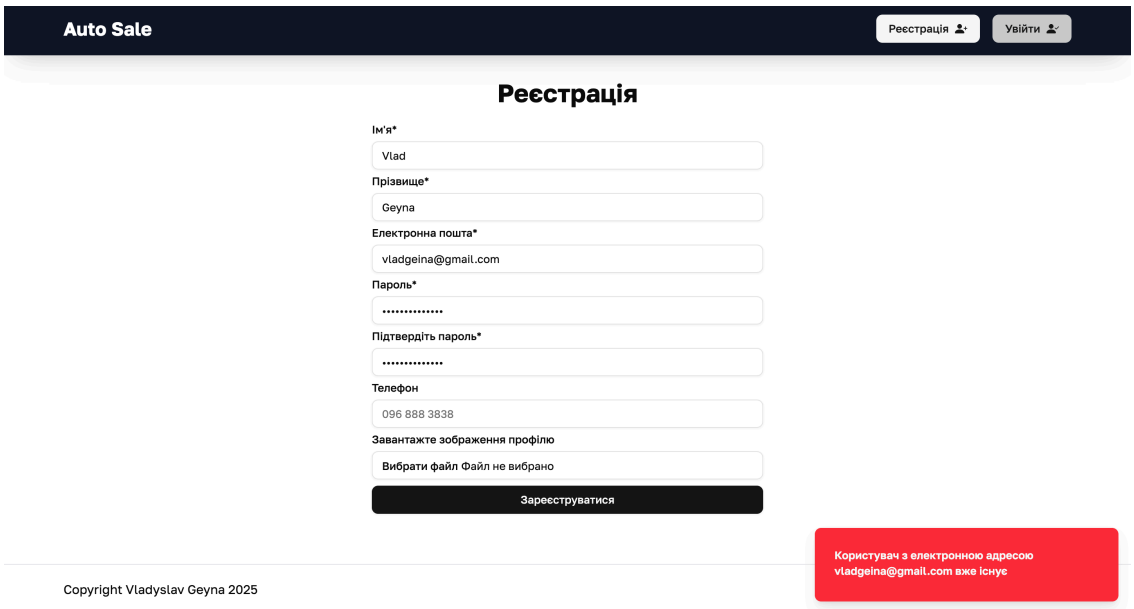


Рис. 3.40 – Помилка при реєстрації

Якщо користувач спробує перейти за посиланням, доступ до якого мають лише адміністратори, або, наприклад, відкрити оголошення, яке було деактивовано продавцем, з’явиться повідомлення про заборону доступу (рис. 3.41).

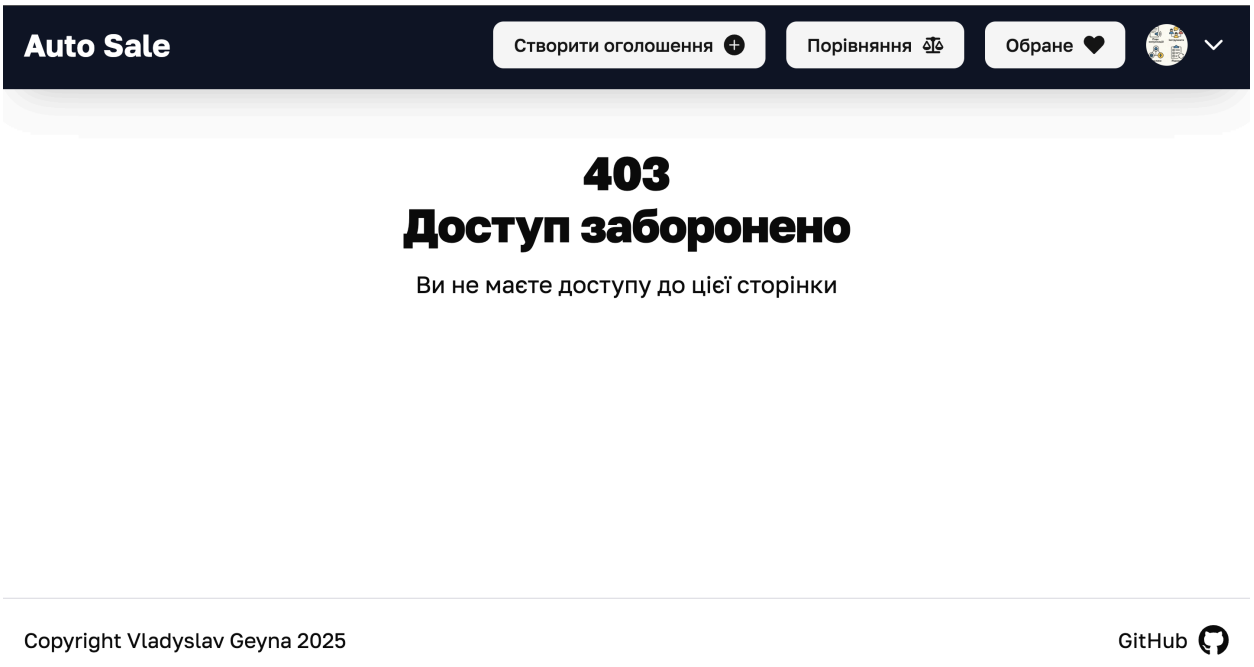


Рис. 3.41 – Доступ заборонено

Також, якщо користувач відкриє неіснуючий URL, система повідомить, що запитуваний ресурс не знайдено (рис. 3.42).

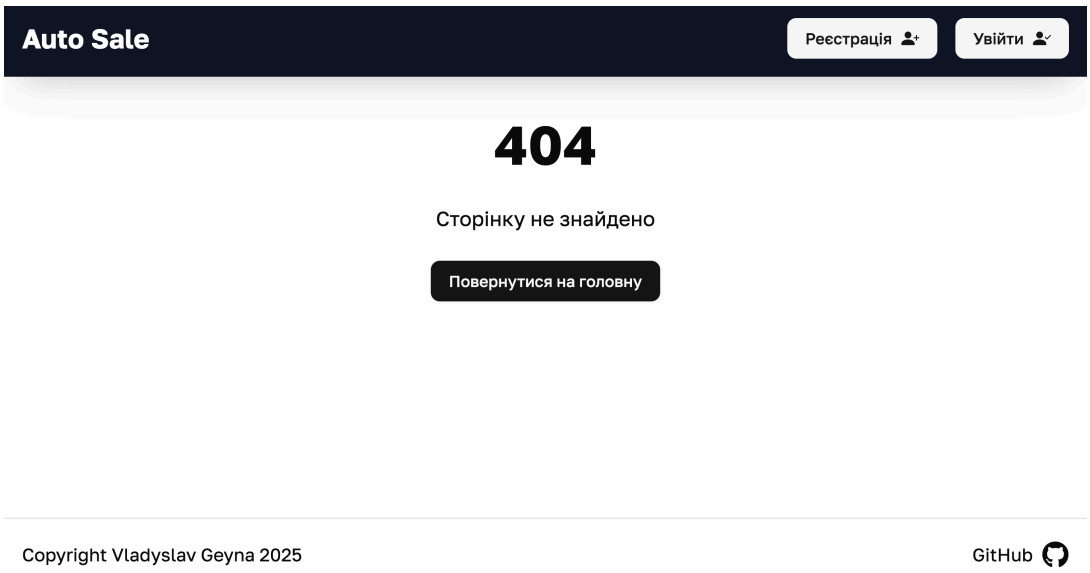


Рис. 3.42 – Ресурс не знайдено

Автомобіль має унікальні поля — VIN-код та номер. Під час створення або редагування оголошення це слід враховувати, оскільки один і той самий автомобіль не може бути виставлений на продаж двічі (рис. 3.43, рис. 3.44).

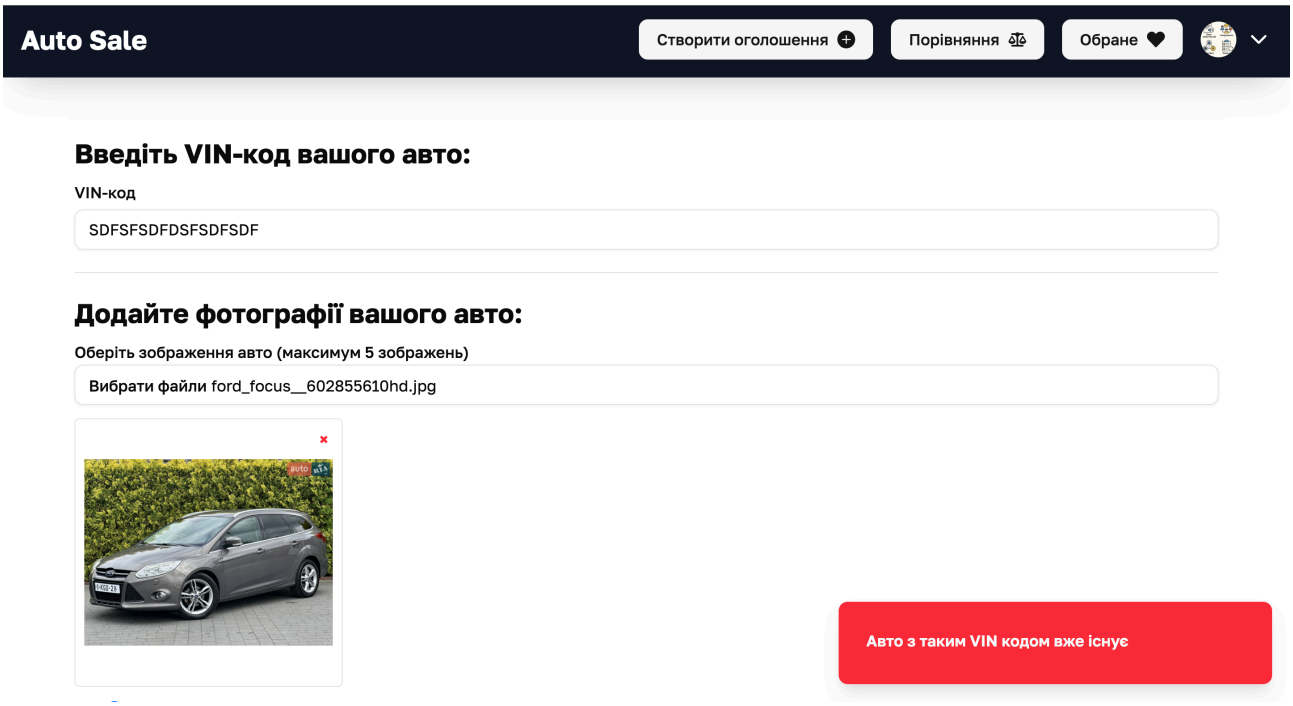


Рис. 3.43 – Помилка при створенні авто з існуючим VIN-кодом

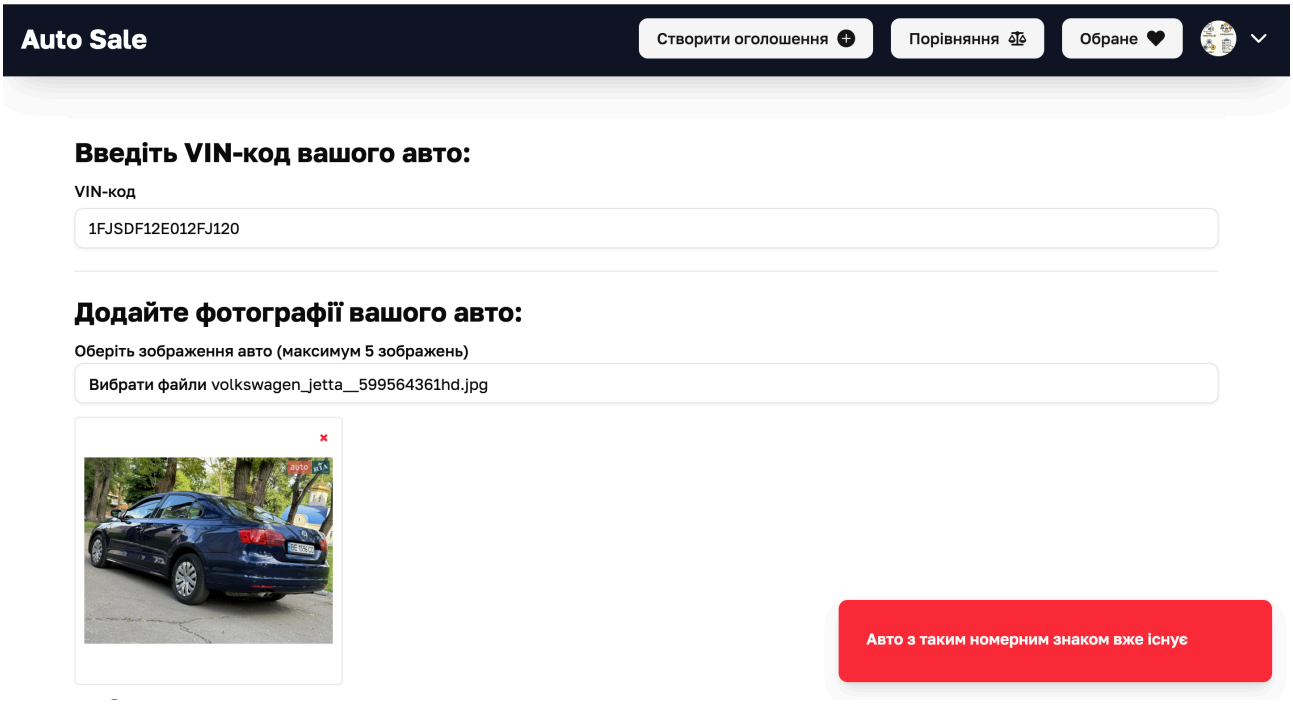


Рис. 3.44 – Помилка при створенні авто з існуючим номером

Оскільки оголошення можна деактивувати, необхідно врахувати це під час відображення у списку “Обране” та в порівнянні — такі оголошення мають автоматично виключатись із цих списків. Крім того, якщо користувача було заблоковано, його оголошення не повинні повертатися в результатах пошуку, а також інші користувачі не повинні мати змоги писати йому в чат.

Для коректного формування рейтингу продавця дозволяється залишати лише один відгук на одного продавця (рис. 3.45).

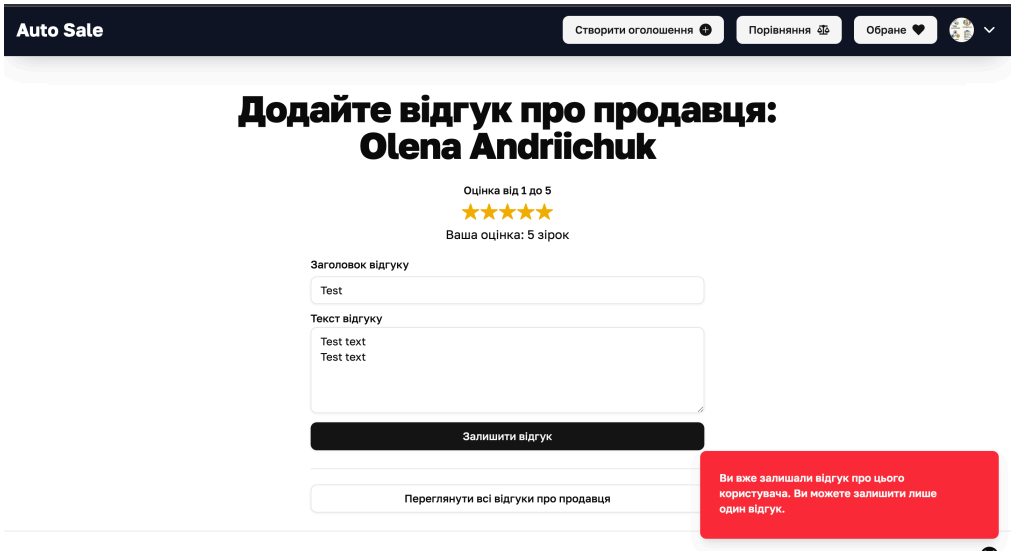


Рис. 3.45 – Помилка при додаванні більше ніж одного відгуку про продавця

Крім того, досвідчений користувач може свідомо змінити дані форми перед відправкою через інструменти розробника або вручну підмінити параметри в URL-рядку. Це становить потенційну загрозу для системи, тому на стороні сервера реалізовано обов’язкову багаторівневу перевірку переданих даних. Зокрема, здійснюється перевірка наявності всіх необхідних полів, правильності та коректності їхніх значень, а також відповідність цим значенням реально існуючих сутностей у базі даних. У випадках, де це доречно, використовується валідація за допомогою регулярних виразів або готових валідаторів — наприклад, для перевірки електронної пошти, номера телефону, дозволених розширень файлів, їх розміру та кількості. Варто зазначити, що поля, які мають приймати лише числові значення, автоматично ігнорують введення будь-яких інших символів. Також кожен запит, який надсилає дані для створення або оновлення чогось у системі, перевіряється ще до того, як потрапить у внутрішню логіку програми. Це дозволяє вчасно відхилити потенційно небезпечні або некоректні запити та забезпечити стабільність і безпеку системи.

Також впроваджено авторизаційні перевірки на певні ендпоінти. Навіть якщо користувач спробує змінити щось на клієнтському рівні, сервер перевіряє, чи має він відповідні права доступу до ресурсу або дії. Це стосується не лише ролей, а й, наприклад, того, що користувач може видалити лише власне оголошення. У разі спроби видалення чужого запису система поверне помилку доступу. Такий підхід застосовується й до багатьох інших дій.

Висновки до третього розділу

У третьому розділі було детально розглянуто порядок встановлення та налаштування веб-платформи для розміщення оголошень про продаж автомобілів. Розгортання платформи здійснюється із застосуванням сучасної технології Docker, яка дозволяє ізолювати кожен компонент системи у власному контейнері. Такий підхід забезпечує не лише простоту в оновленні та

масштабуванні окремих сервісів, а й підвищує загальну стабільність роботи платформи, що особливо важливо при навантаженнях у реальному часі.

У рамках розділу було детально проаналізовано ключовий функціонал платформи, зокрема процеси реєстрації та авторизації користувачів, які гарантують безпечний та зручний доступ до системи. Окрему увагу приділено роботі з оголошеннями, включно з усіма аспектами їх створення, редагування та публікації, а також механізмам модерації, що забезпечують контроль якості та відповідність контенту встановленим правилам. Особливе значення має реалізація системи реального часу — чату, що дає змогу користувачам швидко та ефективно спілкуватися безпосередньо у межах платформи, підвищуючи її інтерактивність та користувацький досвід.

Окремо розглянуто адміністративну панель, яка виконує важливу роль у керуванні платформою. Вона надає адміністраторам інструменти для управління користувачами та оголошеннями, що дозволяє контролювати контент і забезпечувати безпеку системи. Такий централізований підхід до адміністрування сприяє оперативному реагуванню на порушення та підтримці високих стандартів якості платформи.

Крім того, було проведено комплексне тестування основних компонентів системи. Воно підтвердило коректність валідації введених даних, ефективність контролю доступу користувачів до різних ресурсів і стабільність роботи API за різних сценаріїв використання. Це дозволяє з упевненістю стверджувати, що платформа готова до експлуатації в реальних умовах та здатна надійно обслуговувати велику кількість користувачів.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було успішно розроблено сучасну веб-платформу для розміщення оголошень про продаж автомобілів, що відповідає актуальним вимогам як з точки зору функціональності, так і технічної реалізації. Ретельний аналіз існуючих аналогів дозволив виявити їхні переваги та недоліки, що стало основою для формування чітких технічних і функціональних вимог до проектованої системи. На основі цього було обґрунтовано вибір архітектури, технологічного стеку та інструментів розробки, що забезпечують надійність, масштабованість і гнучкість платформи.

Серверна частина системи реалізована у вигляді модульного моноліту на базі Express.js, що дозволяє організувати код у зручній структурі, легко підтримувати та розширювати функціонал. Клієнтська частина побудована з використанням React та TanStack Query, що дає змогу ефективно обробляти асинхронні запити до API, підтримувати кешування даних і забезпечувати високу швидкодію інтерфейсу. Обрана база даних PostgreSQL відповідає вимогам збереження складних зв’язків між об’єктами, що критично важливо для системи з великою кількістю пов’язаних даних, таких як оголошення, користувачі, повідомлення, автомобілі та ін.

У роботі було детально опрацьовано функціональні та нефункціональні вимоги до платформи, зокрема вимоги до продуктивності, безпеки, зручності користування та масштабованості. Особливу увагу приділено ролям користувачів і їхнім правам, що забезпечує належний рівень доступу та персоналізації. Візуалізація бізнес-логіки за допомогою діаграм активностей і класів сприяла кращому розумінню структури та процесів системи, що є важливим етапом при проектуванні складних програмних комплексів.

Реалізація ключових функціональних модулів платформи включала організацію безпечної реєстрації та авторизації користувачів із застосуванням двох JWT-токенів, що дозволяє ефективно контролювати сесії та забезпечує захист від несанкціонованого доступу. Оптимізація роботи з даними досягнута

через впровадження механізмів посторінкового завантаження, фільтрації, серверного кешування за допомогою Redis, а також клієнтського кешування і підтримки оптимістичних оновлень із використанням TanStack Query. Це забезпечує високу швидкодію платформи та знижує навантаження на сервер у умовах великої кількості користувачів і запитів.

Важливою складовою проекту стала система реального часу, зокрема чат, що дозволяє користувачам швидко обмінюватися повідомленнями безпосередньо на платформі, підвищуючи її інтерактивність та привабливість для користувачів. Для адміністрування платформи була розроблена зручна панель управління, яка дає змогу адміністраторам контролювати користувачів, оголошення та інший контент, що сприяє підтримці високих стандартів якості та безпеки.

Процес розгортання та підтримки платформи організовано із застосуванням сучасних технологій контейнеризації на базі Docker, що забезпечує ізоляцію окремих сервісів, спрощує оновлення, масштабування та обслуговування системи. Такий підхід є актуальним і відповідає сучасним вимогам до інфраструктури веб-додатків.

Комплексне тестування розробленої платформи включало перевірку коректності валідації даних, функціональності API, контролю доступу та стабільності роботи у різних сценаріях використання. Результати тестування підтвердили готовність системи до експлуатації, її надійність і здатність обслуговувати велику кількість користувачів з високим рівнем продуктивності.

Отже, виконана робота охоплює всі основні етапи розробки складної веб-платформи — від аналізу і проектування до впровадження та тестування. Розроблена система є масштабованою, безпечною, зручною у використанні та адаптованою до потреб сучасного ринку оголошень про продаж автомобілів. Отримані результати мають практичну цінність і можуть бути використані для подальшого розвитку функціоналу та покращення платформи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. AUTO.RIA [Електронний ресурс] – Режим доступу до ресурсу: https://auto.ria.com/uk/ .
2. RST [Електронний ресурс] – Режим доступу до ресурсу: https://rst.ua/ukr/ .
3. OLX [Електронний ресурс] – Режим доступу до ресурсу: https://www.olx.ua/uk/ .
4. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/ .
5. Офіційна документація Docker, для налаштування та розгортання контейнерів [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/.
6. Express.js [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/ .
7. PostgreSQL documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/.
8. Gettings started with TypeORM [Електронний ресурс] – Режим доступу до ресурсу: https://typeorm.io/docs/getting-started.
9. SQL Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/sql/.
10. FormData [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/FormData.
11. Multer [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/en/resources/middleware/multer.html.
12. Reusing Logic with Custom Hooks [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/learn/reusing-logic-with-custom-hooks.
13. Інструкція щодо використання TanStack Query [Електронний ресурс] – Режим доступу до ресурсу: https://tanstack.com/query/latest/docs/react/overview.
14. Cross-Origin Resource Sharing [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS.

15. Getting Started with Axios [Електронний ресурс] – Режим доступу до ресурсу: https://axios-http.com/docs/intro.

16. Офіційна документація React.js [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/.

17. Amazon Simple Storage Service Documentaion [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/s3/.

18. OAuth 2.0 для доступу до Google API [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/identity/protocols/oauth2.

19. Інструкція щодо використання TanStack Query [Електронний ресурс] – Режим доступу до ресурсу: https://tanstack.com/query/latest/docs/react/overview.

20. WebSockets with Socket.IO [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/learn-websockets-socket-io/.

21. Socket.IO documentation [Електронний ресурс] – Режим доступу до ресурсу: https://socket.io/docs/v4/.

22. Learn REST API design [Електронний ресурс] – Режим доступу до ресурсу: https://www.restapitutorial.com/.

23. Nodemailer documentation [Електронний ресурс] – Режим доступу до ресурсу: https://nodemailer.com/.

24. Redis Quick Start [Електронний ресурс] – Режим доступу до ресурсу: https://redis-doc-test.readthedocs.io/en/latest/topics/quickstart/.

25. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/introduction.

26. Shadcn/ui component library documentation [Електронний ресурс] – Режим доступу до ресурсу: https://ui.shadcn.com/docs.

27. Як будувати UML-діаграми [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/40575/.

28. Що таке тестування ПЗ, його етапи, види, інструменти [Електронний ресурс] – Режим доступу до ресурсу: https://university.sigma.software/what-is-software-testing/.

29. Як будувати UML-діаграми [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/40575/.

30. Аналіз вимог [Електронний ресурс] – Режим доступу до ресурсу: https://visuresolutions.com/uk/alm-guide/requirement-analysis.

31. Що таке функціональні вимоги: приклади, визначення [Електронний ресурс] – Режим доступу до ресурсу: https://visuresolutions.com/uk/alm-guide/functional-requirements.

32. Нефункціональні вимоги: приклади, типи, підходи [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/non-functional-requirements-examples-types-approaches/.

33. Plate Recognizer Snapshot API [Електронний ресурс] – Режим доступу до ресурсу: https://guides.platerecognizer.com/docs/snapshot/api-reference/.

34. CarQuery API Usage [Електронний ресурс] – Режим доступу до ресурсу: https://www.carqueryapi.com/documentation/api-usage/.

35. What is middleware? [Електронний ресурс] – Режим доступу до ресурсу: https://www.redhat.com/en/topics/middleware/what-is-middleware.

36. Get started with Tailwind CSS [Електронний ресурс] – Режим доступу до ресурсу: https://tailwindcss.com/docs/installation/using-vite.

37. Get started with draw.io [Електронний ресурс] – Режим доступу до ресурсу: https://www.drawio.com/doc/.

38. High performance Node.js image processing [Електронний ресурс] – Режим доступу до ресурсу: https://sharp.pixelplumbing.com/.

39. Passport.js documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.passportjs.org/docs/.

ДОДАТКИ

Додаток А

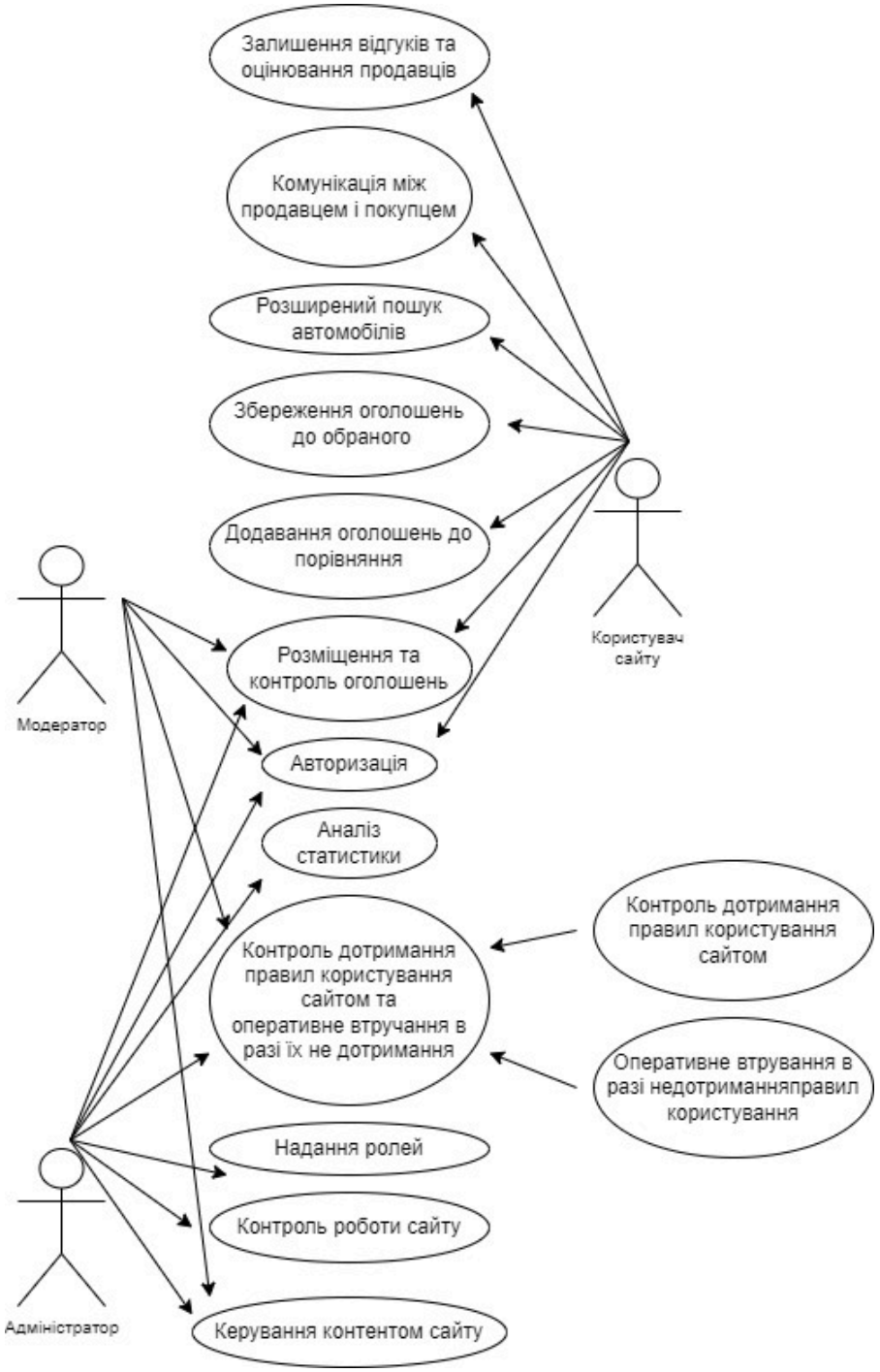


Рис. А.1 – Діаграма варіантів використання

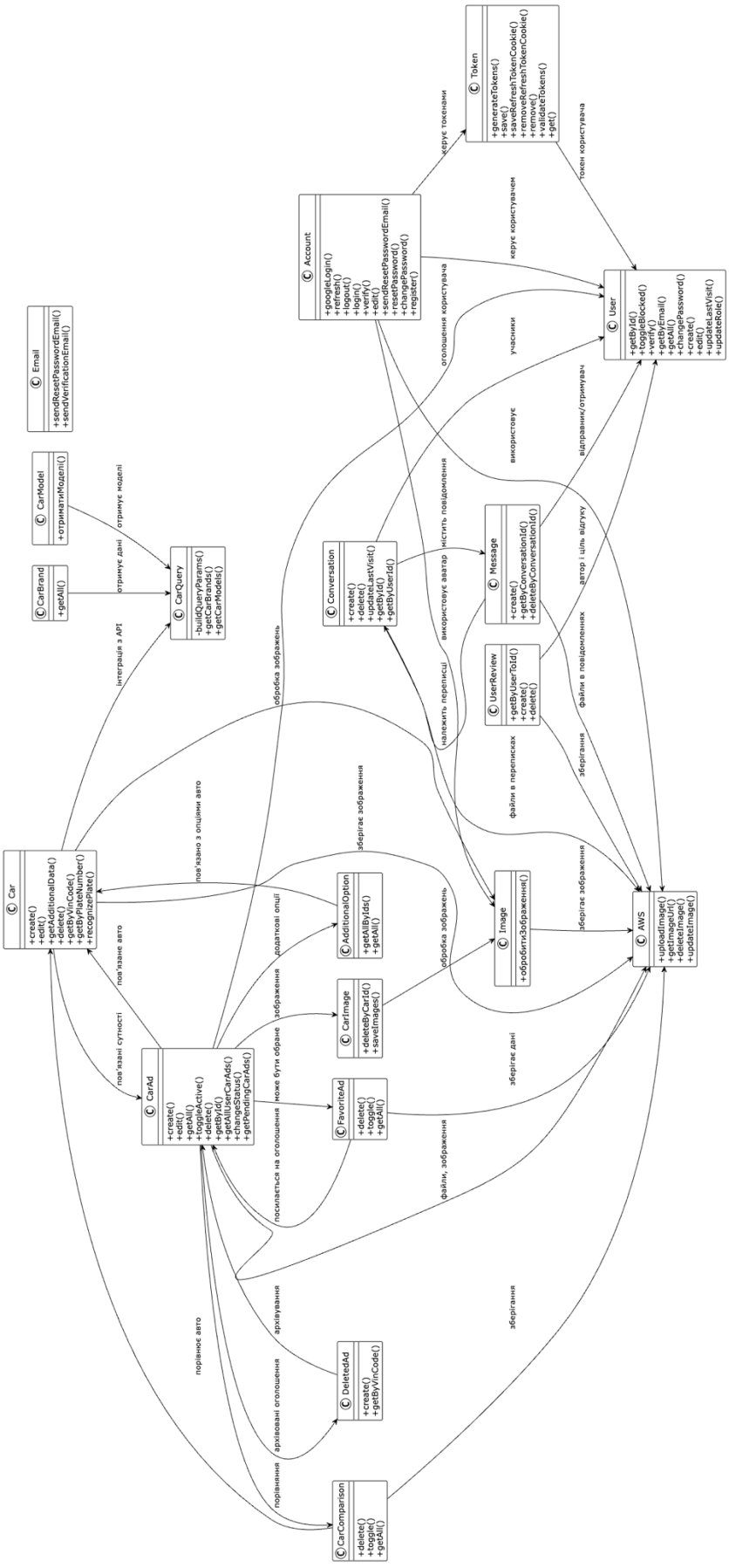


Рис. Б.1 – Діаграма класів

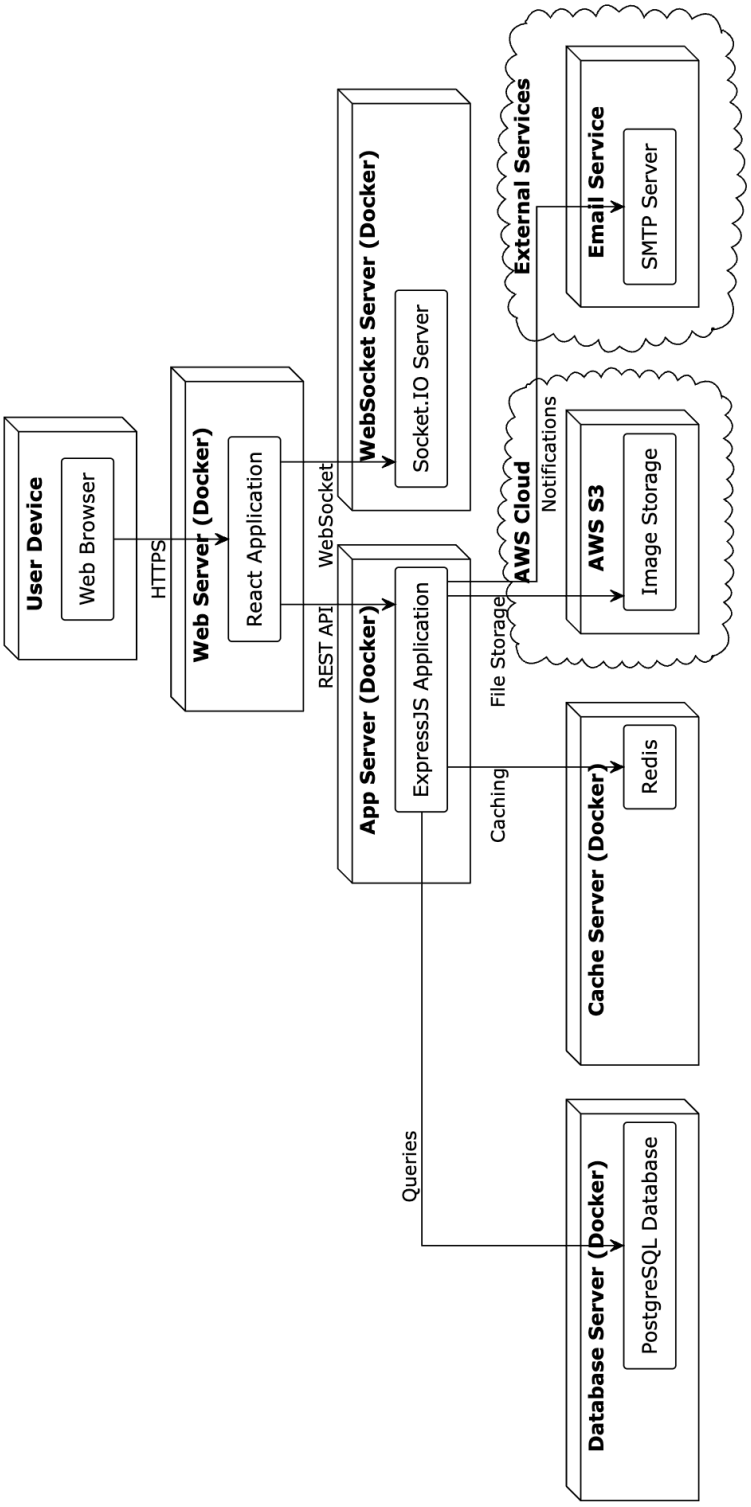


Рис. В.1 – Діаграма послідовності процесу взаємодії з токенами

Додаток Г

Приклад реалізації оптимістичного оновлення кешу під час видалення оголошення:

```
const deleteCarAd = async (carAdId: number) => {
  await authApi.delete(`${URL}/${carAdId}`);
};
export const useDeleteCarAd = (
  carAdId: number,
  options?: UseMutationOptions<void, ApiError, number, unknown>
) => {
  const queryClient = useQueryClient();
  const { authUser } = useContext(AuthUserContext);
  const optimisticCarAdQueryKey = [CAR_ADS, carAdId];
  const optimisticCarAdsByUserIdQueryKey = [CAR_ADS, authUser?.id];
  const optimisticPendingCarAdsQueryKey = [PENDING_CAR_ADS];
  return useMutation({
    ...options,
    mutationKey: [DELETE_CAR_AD, carAdId],
    mutationFn: deleteCarAd,
    onMutate: async () => {
      await queryClient.cancelQueries({ queryKey: optimisticCarAdQueryKey });
      await queryClient.cancelQueries({
        queryKey: optimisticCarAdsByUserIdQueryKey,
      });
      await queryClient.cancelQueries({
        queryKey: optimisticPendingCarAdsQueryKey,
      });
      const previousCarAd = queryClient.getQueryData<GetCarAdByIdResponse>(
        optimisticCarAdQueryKey
      );
      const previousCarAdsByUserId =
        queryClient.getQueryData<GetCarAdsByUserIdResponse>(
          optimisticCarAdsByUserIdQueryKey
        );
      const previousPendingCarAds =
        queryClient.getQueryData<GetPendingCarAdsByResponse>(
          optimisticPendingCarAdsQueryKey
        );
      if (previousCarAd)
        queryClient.removeQueries({
          queryKey: optimisticCarAdQueryKey,
        });
      if (previousCarAdsByUserId)
        queryClient.setQueryData<GetCarAdsByUserIdResponse>(
          optimisticCarAdsByUserIdQueryKey,
          (prevCarAds) =>
            prevCarAds
              ? {
                  count: prevCarAds.count - 1,
                  carAds: prevCarAds.carAds.filter(
                    (carAd) => carAd.id !== carAdId
                  ),
                }
              : prevCarAds
        );
      if (previousPendingCarAds)
        queryClient.setQueryData<GetPendingCarAdsByResponse>(
```

```
  optimisticPendingCarAdsQueryKey,
          (prevPendingCarAds) =>
            prevPendingCarAds
              ? {
                  count: prevPendingCarAds.count - 1,
                  carAds: prevPendingCarAds.carAds.filter(
                    (carAd) => carAd.id !== carAdId
                  ),
                }
              : prevPendingCarAds
        );
      return {
        previousCarAd,
        previousCarAdsByUserId,
        previousPendingCarAds,
      };
    },
    onError: (...props) => {
      options?.onError?.(...props);
      const [, , context] = props;
      if (context?.previousCarAd) {
        const { previousCarAd } = context;
        queryClient.setQueryData(optimisticCarAdQueryKey, previousCarAd);
      }
      if (context?.previousCarAdsByUserId) {
        const { previousCarAdsByUserId } = context;
        queryClient.setQueryData(
          optimisticCarAdsByUserIdQueryKey, previousCarAdsByUserId);
      }
      if (context?.previousPendingCarAds) {
        const { previousPendingCarAds } = context;
        queryClient.setQueryData(
          optimisticPendingCarAdsQueryKey,
          previousPendingCarAds
        );
      }
    },
    onSettled: (...props) => {
      options?.onSettled?.(...props);
      const [, , , context] = props;
      const { previousCarAd } = context ?? {};
      queryClient.invalidateQueries({ queryKey: optimisticCarAdQueryKey });
      queryClient.invalidateQueries({
        queryKey: [CAR_ADS],
      });
      queryClient.invalidateQueries({
        queryKey: optimisticCarAdsByUserIdQueryKey,
      });
      queryClient.invalidateQueries({
        queryKey: optimisticPendingCarAdsQueryKey,
      });
      queryClient.invalidateQueries({
        queryKey: [USERS],
      });
      queryClient.invalidateQueries({
        queryKey: [DELETED_ADS, previousCarAd?.vinCode],
      });
    },
  });
};
```

Лістинг коду індексного файлу додатку Express.js:

```
class App {
  private port: number;
  private app: express.Application;
  private URI_PREFIX: string = "api";
  constructor(port: number) {
    this.port = port;
    this.app = express();
    this.initializeMiddlewares();
    this.initializeRoutes();
    this.initializeErrorHandling();
  }
  private getRouteUri(resourceName: string) {
    return `/${this.URI_PREFIX}/${resourceName}`;
  }
  private initializeRoutes() {
    this.app.use(this.getRouteUri("account"), accountRouter);
    this.app.use(this.getRouteUri("car-ads"), carAdRouter);
    this.app.use(this.getRouteUri("cars"), carRouter);
    this.app.use(this.getRouteUri("car-models"), carModelRouter);
    this.app.use(this.getRouteUri("car-brands"), carBrandRouter);
    this.app.use(this.getRouteUri("car-comparisons"), carComparisonRouter);
    this.app.use(this.getRouteUri("favorite-ads"), favoriteAdRouter);
    this.app.use(this.getRouteUri("conversations"), conversationRouter);
    this.app.use(this.getRouteUri("messages"), messageRouter);
    this.app.use(this.getRouteUri("user-reviews"), userReviewRouter);
    this.app.use(this.getRouteUri("users"), userRouter);
    this.app.use(this.getRouteUri("deleted-ads"), deletedAdRouter);
    this.app.use(
      this.getRouteUri("additional-options"),
      additionalOptionRouter
    );
  }
  private initializeMiddlewares() {
    this.app.use(cookieParser());
    this.app.use(express.json());
    this.app.use(
      session({
        secret: String(process.env.SESSION_SECRET),
        resave: false,
        saveUninitialized: true,
      })
    );
    this.app.use(passport.initialize());
    this.app.use(passport.session());
    this.app.use(
      cors({
        credentials: true,
        origin: process.env.CLIENT_URL,
      })
    );
  }
  private initializeErrorHandling() {
    this.app.use(errorMiddleware);
  }
  public async start() {
    try {
```

```
  await appDataSource.initialize();
      await seedCarOptions(appDataSource);
      this.app.listen(this.port, () => {
        console.log(`Server started on port ${this.port}`);
      });
    } catch (error) {
      console.log(error);
    }
  }
}
```

Лістинг коду сервісу по роботі із зображеннями:

```
class ImageService {
  private imageRepository: Repository<Image>;
  private MAX_SIZE_TO_RESIZE = 1280;
  private COMPRESSION_QUALITY = 80;
  private IMAGE_MIME_TYPE = "image/jpeg";
  constructor() {
    this.imageRepository = appDataSource.getRepository(Image);
  }
  private generateUniqueFileName() {
    return uuid.v4();
  }
  async save(file: Express.Multer.File): Promise<Image> {
    const createdImage = await this.storingImageProcessing(file.buffer);
    return createdImage;
  }
  async update(oldImageId: number, newFile: Express.Multer.File) {
    const oldImage = await this.getById(oldImageId);
    if (!oldImage) {
      throw HttpError.NotFound("Зображення не знайдено");
    }
    const uniqueFileName = this.generateUniqueFileName();
    const buffer = await this.resizeImage(newFile.buffer);
    const fileData: UploadFileInputDto = {
      fileName: uniqueFileName,
      fileBuffer: buffer,
      mimeType: this.IMAGE_MIME_TYPE,
    };
    await awsService.updateImage(oldImage.name, fileData);
    oldImage.name = uniqueFileName;
    const updatedImage = await this.imageRepository.save(oldImage);
    return updatedImage;
  }
  async saveFromUrl(url: string) {
    const res = await fetch(url);
    const bufferArray = await res.arrayBuffer();
    const createdImage = await this.storingImageProcessing(
      Buffer.from(bufferArray)
    );
    return createdImage;
  }
  private async storingImageProcessing(imageBuffer: Buffer) {
    const uniqueFileName = this.generateUniqueFileName();
    const buffer = await this.resizeImage(imageBuffer);
    const fileData: UploadFileInputDto = {
      fileName: uniqueFileName,
      fileBuffer: buffer,
      mimeType: this.IMAGE_MIME_TYPE,
    };
 await awsService.uploadImage(fileData);
```

```
    const image = this.imageRepository.create({
      name: uniqueFileName,
    });
    const newImage = await this.imageRepository.save(image);
    return newImage;
  }
  private async resizeImage(imageBuffer: Buffer) {
    const buffer = await sharp(imageBuffer)
      .resize({
        width: this.MAX_SIZE_TO_RESIZE,
        height: this.MAX_SIZE_TO_RESIZE,
        fit: "inside",
        withoutEnlargement: true,
      })
      .toFormat("jpeg")
      .jpeg({ quality: this.COMPRESSION_QUALITY })
      .toBuffer();

    return buffer;
  }
  getById(id: number) {
    const image = await this.imageRepository.findOneBy({ id });

    return image;
  }
  async deleteByName(imageName: string) {
    await awsService.deleteImage(imageName);

    await this.imageRepository.delete({ name: imageName });
  }
}
```

Лістинг коду WebSocket сервера:

```
const io = new Server({
  cors: {
    origin: process.env.CLIENT_URL,
  },
});
let users = [];
const addUser = (userId, socketId) => {
  users.push({ userId, socketId });
};
const removeUser = (socketId) => {
  users = users.filter((user) => user.socketId !== socketId);
};
const getUsersByUserId = (userId) => {
  return users.filter((user) => user.userId === userId);
};
io.on("connection", (socket) => {
  socket.on("join", (userId) => {
    addUser(userId, socket.id);
  });
  socket.on("checkOnline", (userIdToCheck) => {
    const isOnline = users.some((user) => user.userId === userIdToCheck);
    socket.emit("onlineStatus", { userId: userIdToCheck, online: isOnline });
  });
  socket.on("sendMessage", ({ senderId, receiverId, text }) => {
    const usersWithSameId = getUsersByUserId(receiverId);
    usersWithSameId.forEach((user) => {
      io.to(user.socketId).emit("getMessage", {
```

```
 senderId,
        text,
      });
    });
  });
  socket.on("typing", ({ receiverId }) => {
    const usersWithSameId = getUsersByUserId(receiverId);
    usersWithSameId.forEach((user) => {
      io.to(user.socketId).emit("responseTyping");
    });
  });
  socket.on("disconnect", () => {
    const disconnectedUser = users.find((user) => user.socketId === socket.id);
    removeUser(socket.id);
    if (disconnectedUser) {
      socket.emit("userOffline", { userId: disconnectedUser.userId });
    }
  });
});
io.listen(process.env.SOCKET_PORT);
```