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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для надання послуг лізингу автомобілів»

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

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

_____ Володимир КОРНЄЄВ
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Володимир КОРНЄЄВ

Керівник: Віктор ЛАВРЕНЕНКО
викладач кафедри ІПЗ

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Корнєєву Володимиру Олексійовичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для надання послуг лізингу автомобілей»
керівник кваліфікаційної роботи викладач кафедри ІПЗ Віктор ЛАВРЕНЕНКО, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи лізингу авто, опис методів лізингу авто, технічна документація з описом бібліотек для надання послуг лізингу транспортного засобу.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Провести аналіз сучасних вимог до веб-застосунків.
2. Здійснити огляд та аналіз існуючих платформ для надання послуг лізингу автомобілів, визначити їх переваги та недоліки.

3. Провести огляд засобів для розробки програмного забезпечення клієнт-серверного застосунку для надання послуг лізингу автомобілів.
4. Сформувати вимоги до розробки клієнт-серверного застосунку.
5. Спроєктувати архітектуру клієнт-серверного застосунку.
6.Розробити та протестувати клієнт-серверний застосунок для надання послуг лізингу автомобілів.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Мапа web-застосунку.
5. Діаграма варіантів використання.
6. Діаграма діяльності (оформлення лізингу).
7. Діаграма класів.
8. Екранні форми.
9. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-16.03.2025	
2	Аналіз та дослідження існуючих аналогів	17.03-23.03.2025	
3	Проаналізувати існуючі методи підбору автомобілів для лізингу, вивчити, як різні алгоритми допомагають користувачам обирати автомобілі, і визначити способи покращення цього процесу.	24.03- 31.03 2025	
4	Провести огляд засобів для розробки програмного забезпечення клієнт-серверного застосунку та сформувати вимоги до розробки програмного забезпечення	01.04-08.04.2025	
5	Спроектувати архітектуру клієнт-серверного застосунку	09.04 - 15.04 2025	
6	Розробити та протестувати клієнт-серверний застосунок для надання послуг лізингу автомобілів.	16.04 - 24.04 2025	

7	Оформлення роботи: вступ, висновки, реферат	25.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Володимир КОРНЄЄВ
(підпис)

Керівник
кваліфікаційної роботи _____ Віктор ЛАВРЕНЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 52 стор., 9 табл., 33 рис., 15 джерел.

Мета роботи – допомога у поширенні послуг , що забезпечить швидкий і зручний доступ до послуг лізингу автомобілів.

Об'єкт дослідження – процес надання послуг лізингу автомобілів та супровід лізингових контрактів.

Предмет дослідження – web-застосунок, що забезпечує організацію та реалізацію послуг з лізингу автомобілів.

Короткий зміст роботи: В роботі проаналізовано предметну галузь та методи надання послуг лізингу автомобілів. Проведено огляд вже існуючих засобів для лізингу авто: ХапайАвто,Vidi,AutoRia . Розроблено web-застосунок та програмно реалізовані основні функціональні можливості, такі як: реєстрація та авторизація, перегляд типів лізингу, фільтрація авто по ціні, порівняння авто за характеристиками, додавання авто в обране, чат з AI-асистентом, оформлення лізингу та перегляд особистих контрактів. Проведено тестування застосунку. В роботі використано фреймворк Django для розробки бекенду та використано JavaScript, HTML, CSS для розробки фронтенду, GeminiAI API для реалізації функціональності чат-асистенту, PostgreSQL для зберігання даних.

Сферою використання застосунку є організація послуг лізингу авто для користувачів.

КЛЮЧОВІ СЛОВА: WEB-ЗАСТОСУНОК, ЛІЗИНГ, АВТО, PYTHON, DJANGO, AI

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 10
ВСТУП 11
1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ НАДАННЯ ПОСЛУГ ЛІЗИНГУ АВТО 13
1.1 Web-застосунок для надання послуг лізингу авто 13
1.2 Специфіка лізингу авто. 14
1.3 Цільова аудиторія 15
1.4 Дослідження існуючих аналогів 16
1.5 Визначення вимог до застосунку 21
2 ЗАСОБИ РЕАЛІЗАЦІЇ 22
2.1 Середовище розробки 22
2.1.2 PyCharm 22
2.2 Мови програмування 23
2.2.1 Python 24
2.2.2 JavaScript 24
2.2.3 HTML, CSS 25
2.3 Веб-фреймворки 26
2.3.1 Django 26
2.4 Система управління базою даних 27
2.5 Система контролю версій 28
3 ПРОЄКТУВАННЯ 31
3.1 Моделювання архітектури web-застосунку 31
3.3.1 Мапа web-застосунку 31
3.3.2 Діаграма варіантів використання 32
3.3.3 Діаграма діяльності 33
3.3.4 Діаграма класів 35
3.3.5 Діаграма компонентів 36
4. ОПИС РОЗРОБКИ ТА ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ 38
4.1 Структура проєкту 38

4.2 Процес розробки реєстрація та авторизації користувачів 39
4.3 Процес розробки створення особистого контракту та перегляду щомісячних платежів. 41
4.4 Процес розробки оформлення лізингу 42
4.5 Процес розробки AI-помічника для вирішення питань користувачів 43
4.6 Клієнтська частина web-застосунку 44
4.7 Тестування web-застосунку 48
ВИСНОВКИ 53
ПЕРЕЛІК ПОСИЛАНЬ 55
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 56
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ 64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення.

IDE – Integrated Development Environment.

API – Application Programming Interface.

URL - Uniform Resource Locator.

HTML – HyperText Markup Language.

CSS – Cascading Style Sheets .

AI - Artificial Intelligence

ВСТУП

Актуальність лізингу автомобілів обумовлена потребою як фізичних осіб, так і підприємств в отриманні транспортних засобів у швидкий спосіб, уникаючи потреби у негайному внесенні значної суми грошей. В умовах економічної нестабільності, росту цін на нові авто та підвищення ставок за банківськими кредитами, лізинг постає як приваблива альтернатива, що дозволяє користуватися автомобілем вже зараз, розподіляючи витрати на тривалий період часу. Це особливо важливо для малого та середнього бізнесу, який активно використовує автотранспорт у своїй операційній діяльності.

Окрім фінансової вигоди, лізинг автомобілів відкриває можливість оновлення автопарку без потреби самостійного продажу старого транспорту, що зменшує адміністративні витрати та супутні ризики. Лізингові компанії часто надають страхування, технічний супровід та сервіс, що додатково підсилює інтерес цієї послуги. Аналізуючи ринок мобільності, зокрема зростання популярності електромобілів та прагнення до більш раціонального використання ресурсів, лізинг автомобілів залишається сучасним, зручним та ефективним рішенням як для бізнесу, так і для приватних осіб.

У 2025 році лізинг авто в Україні продовжує набирати популярності, стаючи все більш привабливим для автолюбителів. Підвищення вартості нових машин та валютні коливання підігріли інтерес до альтернативних методів придбання транспортних засобів. Оренда авто з правом викупу стає все більш популярною серед тих, хто прагне мати новий автомобіль, не відчуваючи на собі фінансового тягаря.

Нові правила та використання цифрових технологій полегшили процес оформлення. Більшість заявок тепер можна подати через інтернет, а розгляд займає лічені години. Для багатьох це є вирішальним аргументом на користь вибору лізингу.

Завдяки цьому мешканці навіть віддалених населених пунктів мають змогу скористатися послугами провідних лізингових фірм. Онлайн-консультації,

електронний підпис та доставка автомобіля додому — усе це стало звичним явищем у 2025 році.[1]

Об'єктом дослідження є процес надання послуг лізингу автомобілів та супровід лізингових контрактів.

Предметом дослідження є розробка web-застосунку, що забезпечує організацію та реалізацію послуг з лізингу автомобілів.

Метою роботи є допомога у поширенні послуг , що забезпечить швидкий і зручний доступ до послуг лізингу автомобілів.

Методи дослідження: в першу чергу було проаналізовано існуючі методи підбору автомобілів для лізингу. Наступним етапом було проаналізувати стек технологій, які будуть відповідати вимогам до програмного забезпечення. Було обрано фреймворк Django для розробки бекенд частини, JavaScript HTML CSS для розробки фронтенду, PostgreSQL як систему бази даних, та GeminiAI API для реалізації AI-чат асистенту.

Наукова новизна роботи визначається наступними функціональними вимогами: вбудований чат з AI асистентом. Web-застосунок виступає як багатогранний набір усіх найбільш необхідних інструментів в одному місці.

Практична значущість результатів полягає в можливості використання реалізованого застосунку для ефективного надання послуг лізингу автомобілів.

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ НАДАННЯ ПОСЛУГ ЛІЗИНГУ АВТО

1.1 Web-застосунок для надання послуг лізингу авто

Web-застосунок для надання послуг лізингу авто- це онлайн-ресурс, що надає користувачам доступ до найсучасніших інструментів та сервісів, необхідних для підбору, оформлення та керування автомобілями, взятими в лізинг. Цей додаток дає змогу пройти весь шлях - від вибору автівки до підписання персональної угоди - в онлайн режимі, без прив'язки до розташування користувача. Головна задача web-застосунку – це забезпечення зручного, швидкого та максимально прозорого процесу отримання автомобіля у лізинг.

Основні компоненти web-застосунку для лізингу авто включають:

– Каталог автомобілів: список, який показує авто в наявності,

– Мультимедійний контент: фото, що сприяють глибшому пізнанню обраного транспортного засобу.

– Інтерактивні інструменти: фільтри, системи порівняння авто та пошук авто за характеристиками, додавання авто в обране та чат з AI помічником.

– Оформлення лізингу: можливість оформити лізинг, та відстежувати статус щомісячних платежів в особистому кабінеті.

Цей web-застосунок виступає ключем до цифрової трансформації лізингового ринку. Він покращує взаємодію для клієнтів, удосконалюючи операційні процеси для користувачів та лізингових компаній.

Основні завдання веб-сервісу для лізингу авто:

1. забезпечити користувачів доступом до великої кількості машин з інформацією про умови та типи лізингу;

2. автоматичний підрахунок щомісячних платежів;

3. поширення знань про переваги лізингу серед користувачів;

4. відстеження та керування лізинговим договором та платежами через особистий кабінет.

Тепер оцінимо переваги та недоліки web-застосунок для надання послуг лізингу авто.

Переваги:

1. Зручність і легкість доступу: користувачі зможуть ознайомитися з усіма автомобілями та оформити заявку у будь-який зручний час для себе.

2. Економія часу: завдяки вбудованим сервісам (фільтри, порівняння авто, та пошук авто за характеристиками), це допомагає прискорити вибір користувача.

3. Прозорість умов: усі умови та типи лізингу детально прописані на сайті.

4. Онлайн оформлення: процес створення персонального контракту повністю онлайн - це зручно та швидко.

5. Візуалізація авто: фото та технічні характеристики допомагають ухвалити важливе рішення, не відвідуючи салон.

Недоліки:

1.Обмежений досвід: користувач не має можливості взяти обране авто на тест драйв , щоб відчути його в реальних умовах.

2.Відсутність індивідуальної консультації: у певних випадках потребується консультація з менеджером по технічним питанням.

3.Обмежений асортимент в порівнянні офлайн салонами: деякі авто можуть бути недоступні через їх унікальність, що обмежує асортимент.

1.2 Специфіка лізингу авто.

Лізинг авто — це довгострокова оренда авто з можливістю подальшої покупки за залишковою ціною. Платежі за лізинг сплачуються регулярно.

Термін договору зазвичай триває від 1-5 років. Весь процес базується на тристоронній угоді між лізингодавцем, який купує автомобіль, постачальником авто та лізингоодержувачем, який їздить на автомобілі та здійснює регулярну оплату. Протягом усього терміну дії договору власником авто залишається

лізингодавець, а по завершенні терміну лізингоодержувач може викупити транспортний засіб або повернути його, залежно від умов договору.

На відміну від кредитування, лізинг нерідко вимагає меншого початкового внеску та має більш комфортні умови укладання угоди. Лізинг гарантує чіткий план витрат і комфорт у використанні, оскільки більшість додаткових зобов'язань бере на себе лізингова компанія [2].

1.3 Цільова аудиторія

Цільова аудиторія автомобільного лізингу охоплює різноманітні групи користувачів, що бажають отримати швидкий,зручний, доступний та фінансово вигідний спосіб користування авто, не використовуючи потреби до купівлі у власність. Ключові аудиторія це:

1. Фізичні особи: приватні особи, котрі бажають мати новий чи надійний автомобіль, уникаючи значних початкових витрат. Це можуть бути молоді спеціалісти, сім'ї, водії, які регулярно змінюють авто, або ж ті, хто не бажає оформлювати кредит.

2. Малий та середній бізнес: підприємства, яким необхідно одне чи декілька авто для власних потреб (доставка, обслуговування, логістика), але котрі не мають бажання або можливості витрачати великі кошти на придбання власного транспорту.

3. Великі компанії: підприємства, що володіють автопарком та вдаються до лізингу як засобу фінансового менеджменту

4. Автофанати, які хочуть часто змінювати авто: для таких людей, лізинг відкриває двері до керування новими автівками раз на декілька років, уникаючи довгострокових фінансових зобов'язань.

5. Іноземці: люди, які живуть в країні тимчасово та не мають наміру ставати власником автомобіля.

1.4 Дослідження існуючих аналогів

Було проведено аналіз трьох аналогів як ХапайАвто, Vidi, і AvtoRia. Основними критеріями програмного забезпечення є зручність використання, функціональність, адаптивність для потреб клієнтів та компаній.

ХапайАвто - онлайн-платформа зосереджена на ринку автомобілів в Україні. Основною метою є спрощення процесу підбору авто для клієнтів шляхом зручного пошуку , інтерактивних фільтрів і прямої комунікації з менеджером. Веб-сайт дає можливість створювати оголошення приватним особам та автосалонам. ХапайАвто показує себе на українському ринку як швидкий та простий онлайн майданчик для пошуку авто в будь-якому місті України.

Переваги ХапайАвто:

- Зрозумілий інтерфейс: зручний для користувача інтерфейс з мінімалістичним дизайном.
- Фокус на вживані авто: платформа допоможе підібрати гарне вживане авто, що найчастіше шукають студенти абе інші користувачі.
- Швидке завантаження сайту: оптимізовано під мобільні пристрої.

Недоліки ХапайАвто:

- Обмежена кількість фільтрів: певні ключові параметри не включені.
- Мала частка офіційних дилерів: орієнтація на приватних продавців, що ускладнює перевірку авто.
- Низький рівень безпеки: відсутні інструменти для перевірки транспортного засобу.

Функціональні можливості:

- Особистий кабінет: базовий для розміщення оголошень та перегляду авто.
- Оформлення лізингу: прямий продаж між продавцем та покупцем.
- Порівняння авто: відсутнє.
- Фільтрація за ціною: базова.

- Додавання в обране: відображається в особистому кабінеті.
- Пошук авто: пошук характеристик за маркою , роком, ціною.

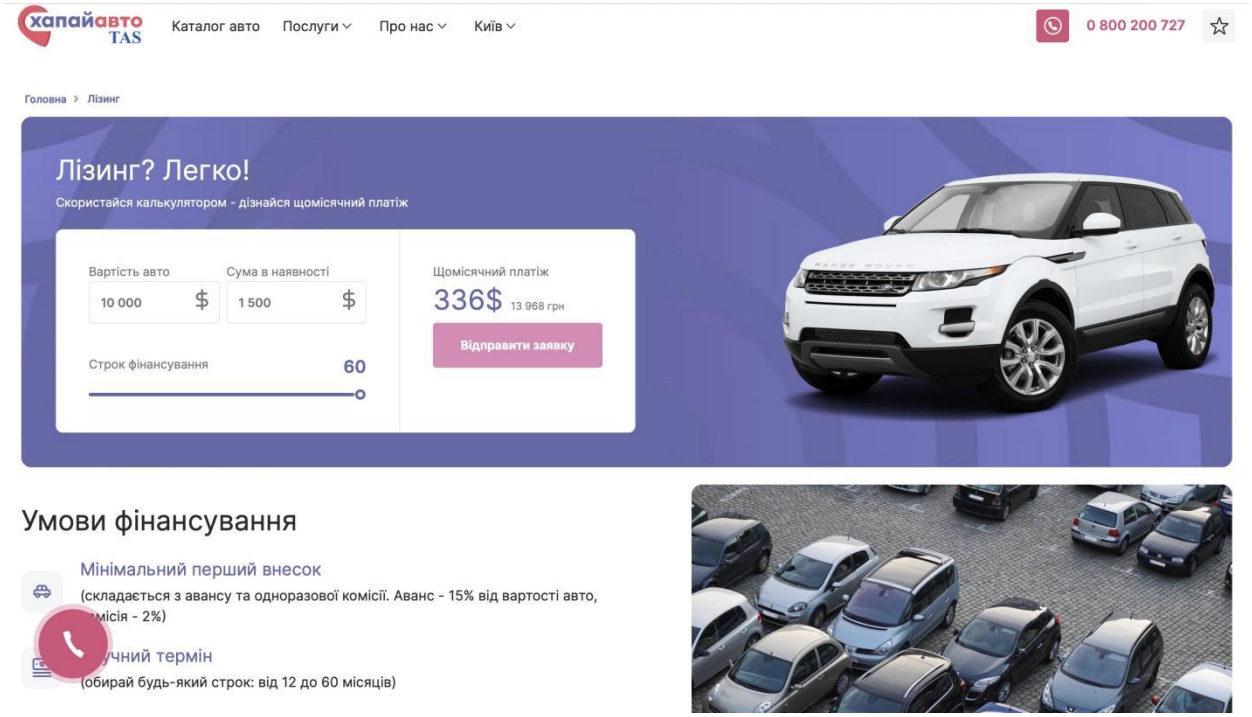


Рис 1.1 Екрані форми веб-сайту ХапайАвто

VidiLeasing- одна з найбільших компаній в Україні, що охоплюй понад 20 офіційних дилерств світових марок. Компанія активно розвиває онлайн напрямки. VidiLeasing платформа, що дозволяє фізичним та юридичним особам оформити лізинг та якісний підбір авто під ключ. Компанія надає повний список послуг від придбання авто до його сервісного обслуговування.

Переваги VidiLeasing:

- Офіційний статус: компанія використовує лише нові автомобілі, гарантуючи високу якість.
- Повний цикл обслуговування: всі питання на рахунок авто такі як лізинг, обслуговування все можна вирішувати в одному місці.
- Фінансові інструменти: можливість швидкого прорахунку лізингу авто.

Недоліки VidiLeasing:

- Обмежена кількість фільтрів: певні ключові параметри не включені.
- Обмежена доступність на ринку: компанія орієнтована на специфічний сегмент ринку, що робить менш доступною для користувачів.
- Відсутня платформа для покупки авто: компанія не пропонує можливості придбати авто через їх сайт.

Функціональні можливості:

- Особистий кабінет: перегляд заявок, статусу лізингу, обслуговування.
- Оформлення лізингу: онлайн-заявка на лізинг.
- Порівняння авто: відсутнє.
- Фільтрація за ціною: часткова, ціни показуються, але без детальної інформації.
- Додавання в обране: відсутнє.
- Пошук авто: пошук характеристик за брендом, моделю, комплектацією.

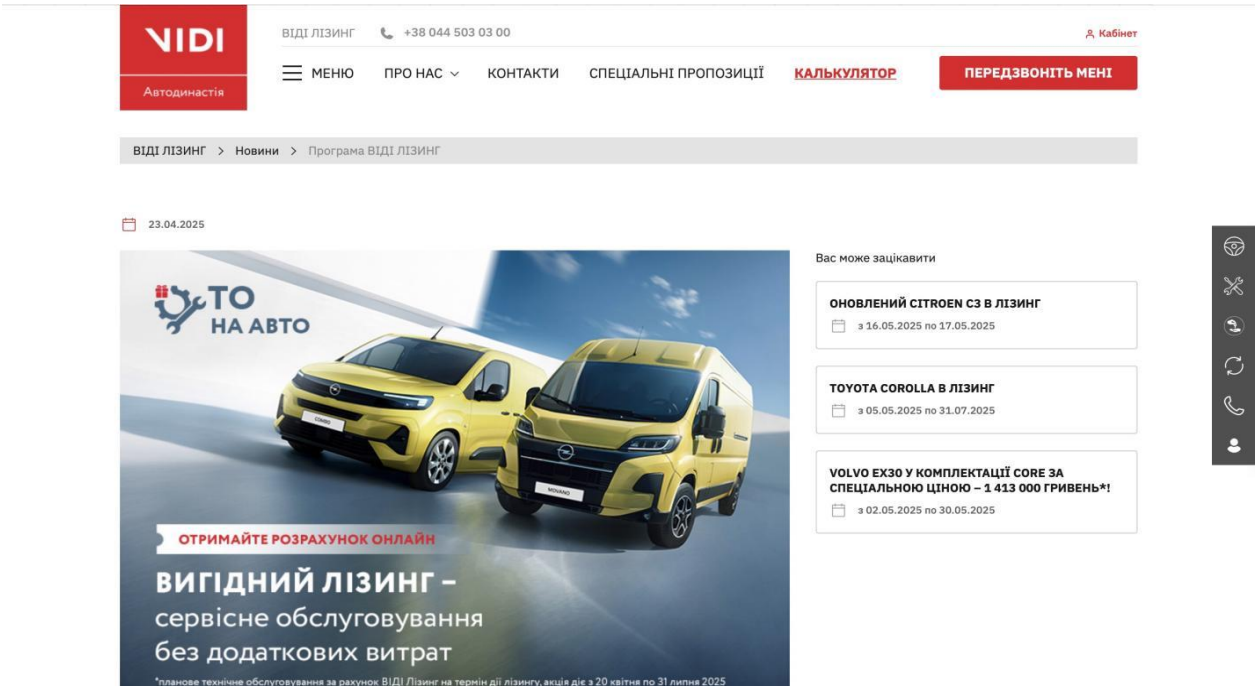


Рис 1.2 Екрані форми веб-сайту VidiLeasing

AutoRia- найпопулярніший веб-застосунок в Україні, який поєднує в собі онлайн-маркетплейс з аналітикою. На сайті розміщуються тисячі оголошень як від приватних осіб та автосалонів. Сервіс пропонує можливості якісного фільтрування, перевірки транспортного засобу за VIN-кодом, відстеження змін цін та також перевірки авто в участі ДТП.

Переваги AutoRia:

- Найбільша база авто в Україні: на сайті можна знайти так як і вживані так і нові авто.
- Якісна система фільтрів: на сайті доступно 30 параметрів пошуку.
- Перевірка VIN-коду: можливість перегляду історії авто.

Недоліки AutoRia:

- Частково платний функціонал: певні сервіси перевірки потребують оплати.
- Наявність фейкових оголошень: трапляються шахрайські оголошення.
- Реклама та дратівливі пропозиції: може заважати пошуку авто.

Функціональні можливості:

- Особистий кабінет: повноцінний для розміщення та редагування оголошень та перегляду авто.
- Оформлення лізингу: доступний лізинг від автосалонів та банків.
- Порівняння авто: інтерактивне порівняння кількох моделей.
- Фільтрація за ціною: розширена включає в себе, сортування за цінами та вибір валюти.
- Додавання в обране: відображається навіть без входу до кабінету.
- Пошук авто: пошук характеристик за маркою , роком, ціною.

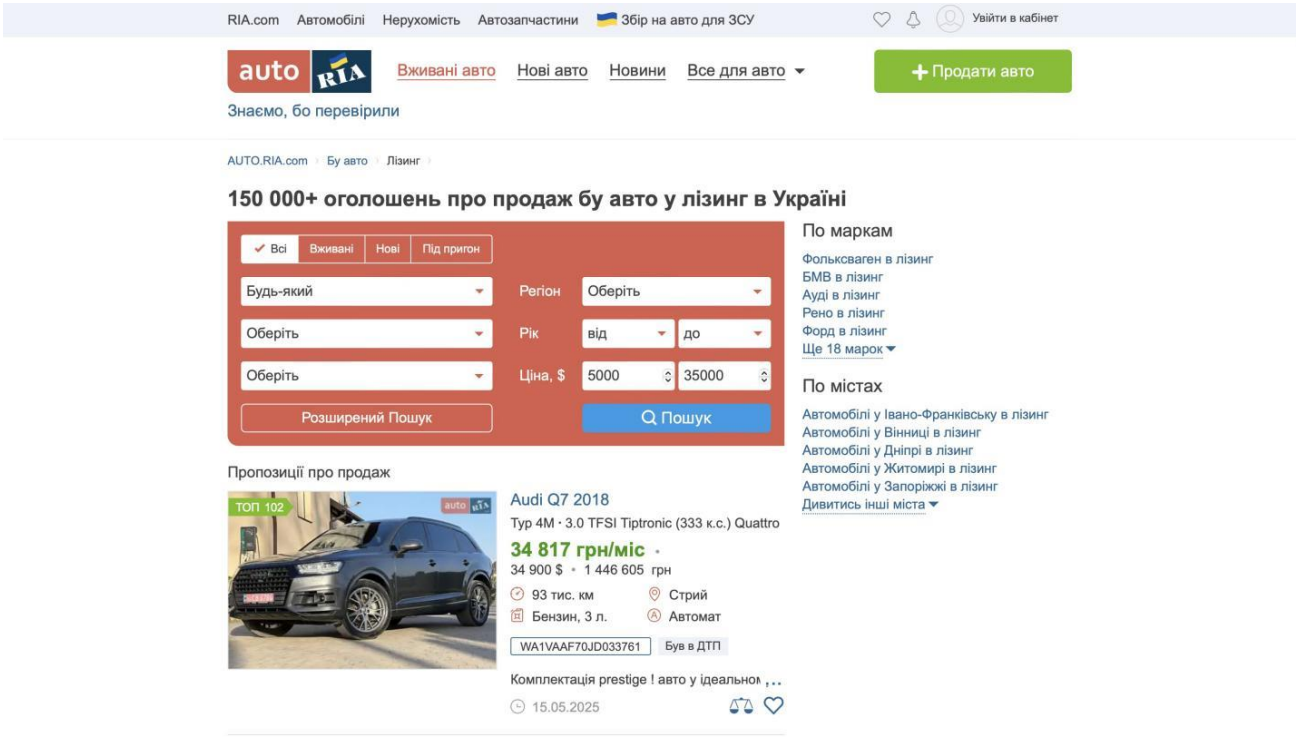


Рис 1.3 Екрані форми веб-сайту AutoRia

1.5 Визначення вимог до застосунку

Зробивши аналіз сутності застосунку, його специфіку, цільову аудиторію та аналогів, було визначено наступні вимоги до застосунку:

1. Авторизація: реєстрація нових користувачів, вхід та вихід із персонального кабінету. Зберігання та перегляд персональних угод.

2. Система пошуку, порівняння, додавання в обране та фільтрації автомобілів: пошук та порівняння авто за характеристиками, фільтрація за ціною, додавання в обране.

3. Сторінка оформлення лізингу: перегляд інформації про авто , та оформлення лізингу за обраним типом.

AI-чат асистент на основі -моделі GeminiAI API: відповіді на питання користувача щодо характеристик авто, типи лізингу та законодавство України про лізинг.

Зведені результати аналізу характеристик додатків для надання послуг лізингу авто наведено в таблиці нижче.

Таблиця 1.1

Критерій	**ХапайАвто**	**VIDI Leasing**	**AutoRia**
Переваги	–Зрозумілий інтерфейс – Фокус на вживані авто –Оптимізовано для мобільних	– Офіційні авто – Повний цикл обслуговування – Фінансові інструменти	– Найбільша база авто – Багато фільтрів – Перевірка VIN-коду
Недоліки	– Обмежені фільтри – Приватні продавці – Відсутність перевірки авто	– Мало фільтрів – Обмежена ринкова доступність – Відсутність онлайн-покупки	– Частково платний функціонал – Фейкові оголошення
Особистий кабінет	Базовий для оголошень та перегляду	Заявки, статус, обслуговування	Повнофункціональний
Оформлення лізингу	Відсутнє	Онлайн-заявка	Через партнерів
Порівняння авто	Відсутнє	Відсутнє	Доступне
Фільтрація за ціною	Базова	Часткова, без деталізації	Повноцінна
Додавання в обране	В особистому кабінеті	Відсутнє	Доступне
Пошук авто	За маркою року і ціною	За брендом моделі комплектації	30+параметрів

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

Інтегроване середовище розробки (IDE) – це програмний засіб, створений для полегшення процесу створення, перевірки та виправлення коду.

IDE включають в себе різноманітні інструменти та функціонал, що дозволяють стандартизувати задачі розробки відповідно до використовуваної мови програмування. Без існування IDE розробнику довелося б самостійно вибирати та управляти всіма інструментами окремо, завдяки середовищу розробки об'єднує всі ці засоби в єдину систему[3].

Рис. 2.1 Логотип інтегрованих середовищ розробки

2.1.2 PyCharm

PyCharm – це інтегроване середовище розробки (IDE) для мови програмування Python, розроблене компанією JetBrains. Воно допомагає Python-програмістам розробляти якісне ПЗ завдяки якісному автодоповненню, підсвічуванню синтаксису та багато інших корисних інструментів[4].

PyCharm допомагає програмістам багатим списком ключових функцій, які значно спрощують процес створення ПЗ. Ось перелік:

1. Налагодження коду: потужний інструмент, який дозволяє виявляти та виправляти вади. Реалізація проходить через встановлення зупинок та дослідження значень змінних.

2. Підтримка систем керування версіями: інтеграція з Git, та схожими системами дозволяє комфортно та якісно взаємодіяти з історією змін.

3. Автодоповнення коду: розумне автодоповнення пришвидшує процес створення ПЗ та зменшує ризик помилок.

4. Перевірка коду: Постійний аналіз коду сприяє виявленню можливих помилок та підтримці високої якості програмного забезпечення.

5. Інтеграція з віртуальним середовищем: PyCharm пропонує комфортний інтерфейс для взаємодії з віртуальними оточеннями Python[5].

Рис. 2.2 Логотип PyCharm

2.2 Мови програмування

Мова програмування — це штучна мова, розроблена для забезпечення передачі інструкцій машинам, зокрема комп'ютерним системам.

Мови програмування використовуються для розробки програм, які допомагають людству вирішувати їхні персональні завдання.

2.2.1 Python

Python - це мова програмування, що працює через інтерпретатор, з можливістю інтерактивного спілкування, яка базується на об’єктах, є високорівневою та підходить для вирішення різноманітних завдань. Вона використовує динамічну, але водночас строгу систему типів даних та автоматичне керування пам'яттю. Головною метою Python є збільшення ефективності праці розробників, покращення читабельності коду та забезпечення легкого перенесення програм на різні платформи. Ідея створення цієї мови виникла в кінці 1980-х років, а безпосередня розробка стартувала в 1989 році, ініціатором став Гвідо ван Россум, співробітник нідерландського інституту CWI.

Мова програмування Python є проста та лаконічна, що робить її більш зрозумілою та досутупною ніж код інших мов програмування.

Python здатний функціонувати на різних типах обладнання, забезпечуючи єдиний інтерфейс незалежно від платформи. Завдяки легкості у вивченні та відсутності потреби у компіляції, розробка програм на Python стає значно швидшою. Код Python зазвичай значно компактніший порівняно з аналогічними програмами, написаними іншими мовами програмування. Це робить його ідеальним для швидкого прототипування в процесі стрімкої розробки програмного забезпечення[6].

Рис. 2.3 Логотип Python

2.2.2 JavaScript

JavaScript - це скриптова мова високого рівня, що активно застосовується для розробки додатків та веб-сайтів. Вона створює веб-сторінки, додаючи

анімацію, спливаючі вікна, перевіряючи форми та оновлюючи контент, роблячи інтерфейс зрозумілим для користувацьких дій. Саме JavaScript робить сайти функціональними та живими.

JavaScript виконується миттєво всередині веб-браузера, без потреби в попередньому перетворенні. Програміст змінює код JavaScript і одразу бачить отриманий результат. JavaScript підтримує фундаментальні парадигми програмування. Завдяки цьому стає можливим конструювання комплексних структур даних, оптимізація організації коду, використання функціональних підходів та детальне окреслення кроків для реалізації поставлених завдань[7].

Рис. 2.4 Логотип JavaScript

2.2.3 HTML, CSS

HTML – це мова розмітки гіпертексту, яка оперує тегами та використовується для організації й структуризації веб сторінок. Якби сайт був будинком, то HTML визначав би розташування кімнат та їх наповнення, проте не відповідає за дизайн. Саме HTML показує браузеру, де має бути заголовок, текст, зображення, посилання та інші елементи. Це основний скелет, на якому зводиться кожен вебсайт. HTML необхідний, аби браузер коректно відображає сторінки.

CSS (таблиці стилів) визначає візуальне оформлення веб-сторінки. Якщо HTML можна вважати основою, подібною до манекена, то CSS – це одяг, який

робить його гарним і стильним. CSS використовує стилі до елементів, створених HTML. Наприклад, змінює колір тексту, розмір шрифту, інтервали між абзацами, додає анімацію та багато іншого[8].

Рис. 2.5 Логотип HTML, CSS

2.3 Веб-фреймворки

Веб-фреймворк - це програмний фреймворк, розроблений для конструювання динамічних веб-сторінок, мережевих програм, служб чи ресурсів. Він полегшує процес створення та звільняє від потреби писати одноманітний код.

2.3.1 Django

Django- безкоштовний та відкритий фреймворк для розробки веб-додатків, створений мовою програмування Python.Створений у 2005 році, Django постав як відповідь на потребу спростити та прискорити процес розробки складних веб-застосунків, орієнтованих на роботу з базами даних.Django використовує філософію DRY ("Не повторюйся"), завдяки чому розробники пишуть менше коду, водночас підвищуючи рівень повторного використання. Це помітно пришвидшує загальний процес розробки.

При створенні веб-сайту, ви зустрінете потребу в низці схожих компонентів: система керування авторизацією (реєстрація, вхід, вихід), форми для введення даних, функціонал завантаження файлів та інше.

Інші розробники давно звернули увагу на ці загальні труднощі, що виникають у процесі веб-розробки. Вони об'єднали зусилля та створили фреймворки (Django – один із них), які пропонують готові до використання складові.

Мета фреймворків – позбавити вас потреби винаходити вже існуючі рішення з нуля та сприяти зменшенню певних витрат часу й ресурсів при розробці нового сайту[9].

Рис. 2.6 Логотип Django

2.4 Система управління базою даних

PostgreSQL – це об'єктно-реляційна система керування базами даних, створена для збереження, обробки та управління значними масивами інформації. Розроблена з акцентом на можливості розширення та відповідність стандарту SQL, PostgreSQL забезпечує підтримку як реляційних моделей, так і окремих рис NoSQL, наприклад, зберігання даних JSON та ключ-значення.

Її головна мета – надати захищене та стабільне рішення для зберігання даних, яке дає змогу користувачам ефективно оперувати складними транзакціями та виконувати аналіз даних у реальному часі.

Основні переваги:

1. Висока сумісність із SQL: PostgreSQL охоплює майже всі функції стандартного SQL, завдяки чому його зручно використовувати для розробки комплексних запитів та транзакцій.

2.Надійність та відповідність ACID-стандартам: підтримка ACID (атомарність, узгодженість, ізоляція, довговічність) забезпечує стійкість та передбачуваність PostgreSQL під час роботи з надважливими даними.

3. Розширюваність: PostgreSQL дає користувачам змогу розширювати його функціональність власними функціями, що є корисним для конкретних задач та пристосування до унікальних потреб.

4.Підтримка JSON і NoSQL: окрім традиційних реляційних баз, PostgreSQL чудово взаємодіє з JSON-даними. Це надає значну перевагу при розробці гібридних застосунків, які поєднують реляційну модель з можливостями документно-орієнтованого зберігання.

Вибір PostgreSQL має зміст, якщо проект вимагає надійну, здатну до масштабування базу даних з розвиненими функціями SQL та адаптивністю до різних типів даних[10].

Рис. 2.7 Логотип PostgreSQL

2.5 Система контролю версій

Git – це система для керування та контролю версіями. Це найпопулярніший та безкоштовний інструмент, в якому зберігається код та історія його змін.

Головними завданнями Git є:

1. Збереження програмного коду разом із його історією змін.

2. Збереження відомостей про користувачів, що вносять зміни в код

3. Маємо здатність повернутися до попередніх версій коду.

4. Реалізація кінцевого продукту до випуску.

Рис. 2.8 Логотип Git

GitHub – це один з багатьох сервісів, що використовують Git. Щоб було зрозуміліше, можна вважати це соціальною платформою для програмістів, де вони можуть бачити ПЗ один одного, брати участь у розробці, залишати коментарі та взаємодіяти між собою таким чином.

GitHub домогає:

1. Завантажувати код.

2. Застосування інструментів для спільної праці.

3. Давати відгук та писати коментарі про роботи інших програмістів.

4. Створення особистих та спільних репозиторіїв для публікування власного ПЗ.

GitHub - це, зокрема, потужний засіб для зберігання репозиторіїв Git, що надає змогу:

1. Співпрацювати з репозиторіями.

2. Коригування помилок.

3. Оптимізувати розробку ПЗ та поширити функціональні можливості.

GitHub — відіграє важливу роль у сучасному IT-світі. Вона допомагає розробникам зручні інструменти для зберігання, контролю версій, документування та командної роботи над проєктами. Використання системи Git, GitHub дозволяє ефективно відстежувати зміни в коді, швидко виявляти помилки та впроваджувати оновлення. Платформа також сприяє розвитку відкритого коду, надаючи можливість брати участь у проєктах з усього світу. Отже, GitHub є ключовою частиною професійного середовища програмістів, що допомагає не лише технічному прогресу, а й формуванню активної роботи[11].

Рис. 2.9 Логотип GitHub

3 ПРОЄКТУВАННЯ

3.1 Моделювання архітектури web-застосунку

Для розуміння як має бути розроблений та як в майбутньому має функціонувати web-застосунок, було створено такі діаграми та схеми:

3.3.1 Мапа web-застосунку

Ця діаграма демонструє структуру web-застосунка, пов'язаного з лізингом автомобілів. На ній показані основні сторінки сайту та їх взаємозв'язок:

1. Головна сторінка: сторінка, з якої можна переходити на інші розділи.
2. Сторінка про типи лізингу: сторінка, за допомогою якої можна ознайомитися про види лізингових послуг.
3. Реєстрація: сторінка, де користувач може створити персональний кабінет.
4. Вхід до персонального кабінету: сторінка авторизації для користувачів.
5. Сторінка персонального кабінету: розділ клієнта про персональні угоди.
6. Сторінка авто в наявності: містить деталі про авто в наявності.

Рис. 3.1 Мапа web-застосунку

3.3.2 Діаграма варіантів використання

Діаграма варіантів використання демонструє взаємодію між користувачами та функціональними можливостями web-застосунку. Дана діаграма показує як система має функціонувати з точки зору користувача, не вдаючись до технічних потреб [12]. Ця діаграма показує функціонал web-застосунку, пов’язаного з лізингом автомобілів. Вона показує основні можливості для користувачів:

1. Оформлення авто в лізинг: головна послуга для користувачів.
2. Пошук авто за характеристиками: фільтрація автомобілів за маркою, об’ємом двигуна та типу двигуна.
3. Використання фільтру ціни для перегляду авто: інструмент, який допомагає аналізувати вартість авто.
4. Порівняння авто: функція, яка допомагає порівнювати авто за характеристиками.
5. Додавання авто в обране: користувач може додати авто, яке йому сподобалось.
6. AI-асистент: допомагає у вирішенні питань про авто, типи лізингу та законодавство України про лізинг.
7. Логін та реєстрація: система для створення персонального кабінету для користувачів.
8. Перегляд контрактів: перегляд особистих угод для щомісячних виплат.

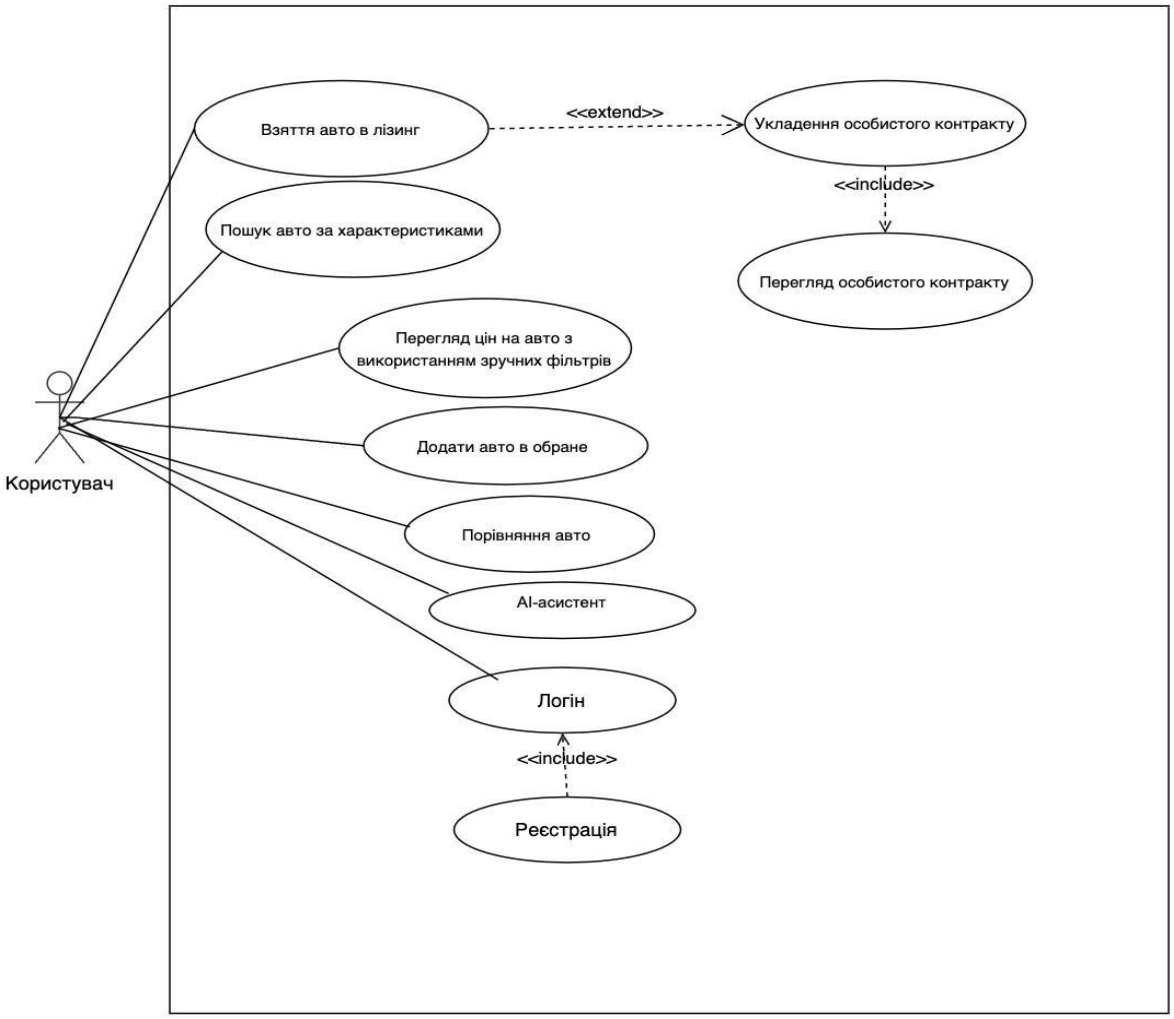


Рис. 3.2 Діаграма варіантів використання

3.3.3 Діаграма діяльності

Діаграма діяльності - це блок-схема, що використовується для візуалізації переходу від однієї дії до іншої. Діяльність показується як робота системи. Головне завдання діаграми - продемонструвати динамічну поведінку системи. Іноді її ще називають об’єктно-орієнтованою блок-схемою[13].

Діаграма діяльності слугує візуальним представленням послідовності операцій, показуючи, як одна дія перетікає в наступну. Вона дає уявлення про роботу системи в цілому та є поглибленим варіантом блок-схеми. Виконавцями дій можуть бути люди, програмні компоненти або комп'ютери. Потік на діаграмі

показує перехід від однієї операції до іншої, та може бути представлений як послідовний, розгалужений, або одночасний.

Ця діаграма описує процес оформлення лізингу автомобіля через особистий кабінет користувача. Вона включає наступні ключові етапи:

1. Перевірка власного кабінету:

- Користувач перевіряє, чи є у нього особистий кабінет.

- Якщо є, користувач вводить свої персональні дані такі як, e-mail та пароль.

2. Реєстрація та авторизація:

- Якщо користувач не має особистого кабінету, повинен зареєструватися.

3. Ознайомлення про умови лізингу:

- Користувач знайомиться та вивчає інформацію про типи лізингу.

4. Вибір авто:

- Вибір авто в наявності.

5. Оформлення лізингу:

- Після вибору авто користувач оформлює обраний тип лізингу.

- Система автоматично створює контракт в особистому кабінеті, для перегляду щомісячних платежів.

Діаграма демонструє простий та швидкий процес оформлення лізингу: від входу в систему до отримання контракту.

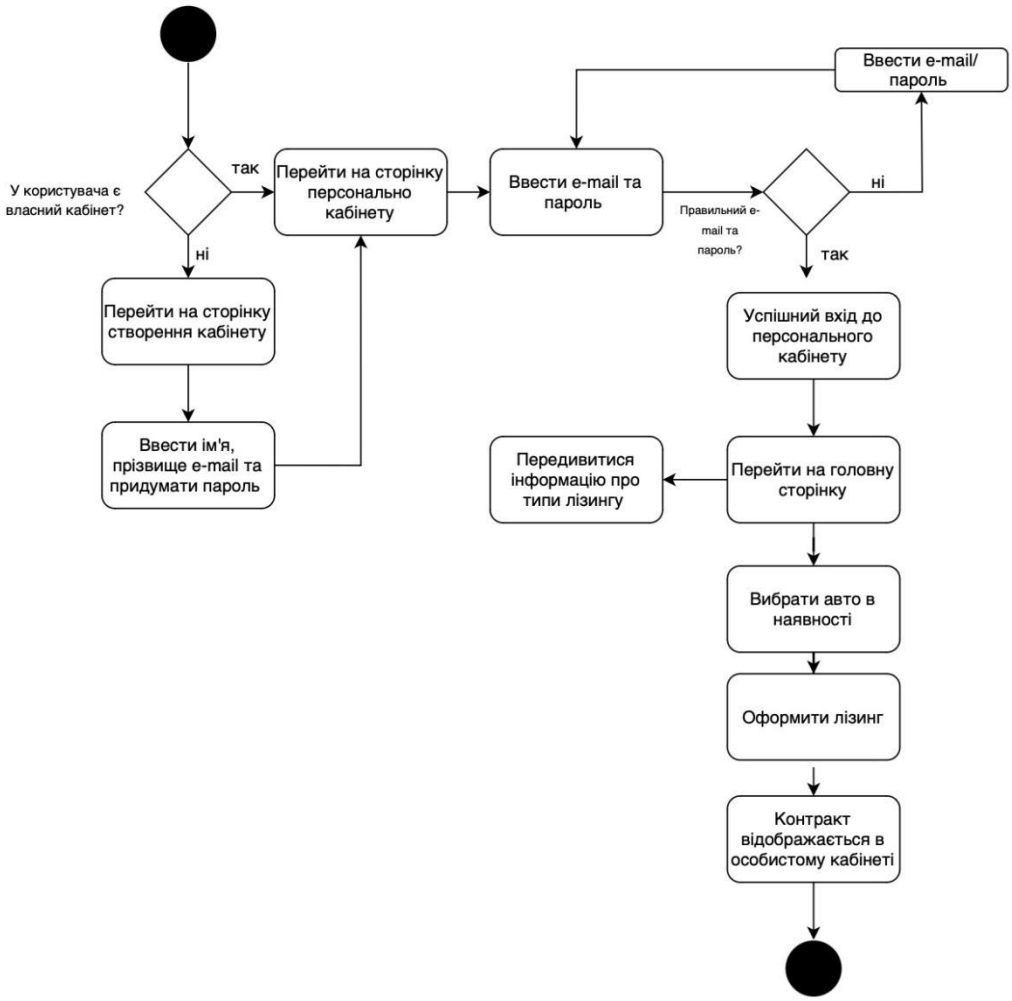


Рис. 3.3 Діаграма діяльності (дія оформлення лізингу)

3.3.4 Діаграма класів

Діаграма класів UML – це візуальний спосіб представлення, що застосовується для створення та зображення об'єктно-орієнтованих систем. Діаграма класів, складова уніфікованої мови моделювання, є статичною структурованою діаграмою, яка показує характеристики системи, класи, операції та взаємозв'язки між об'єктами задля опису архітектури системи. Діаграми класів належать до структурних діаграм, тому що вони визначають, що повинно бути частиною змодельованої системи[14].

Ця діаграма демонструє структуру бази даних для надання послуг лізингу автомобілів. Вона показує основні класи та їх взаємозв'язки:

1. Service Packages (Пакети послуг):

- Складається з назви, опису, ціни, та списку послуг.

- Засоби для підрахунку базових та преміальних послуг.

2. Service (Послуги):

- Демонструє окремі послуги.

3. Car (Авто):

- Складається з деталей авто (назва, ціна , рік, двигун, пробіг, та колір).

- Засоби для розрахунку знижки та перевірки доступності.

4. Leasing (Лізинг):

- Складається з дати, загальної ціни, та шомісячного платежу.

- Пов’язує користувача, автомобілі та пакети послуг.

5. User (Користувач):

- Стандартні поля для персонального кабінет ((ім'я, email, пароль).

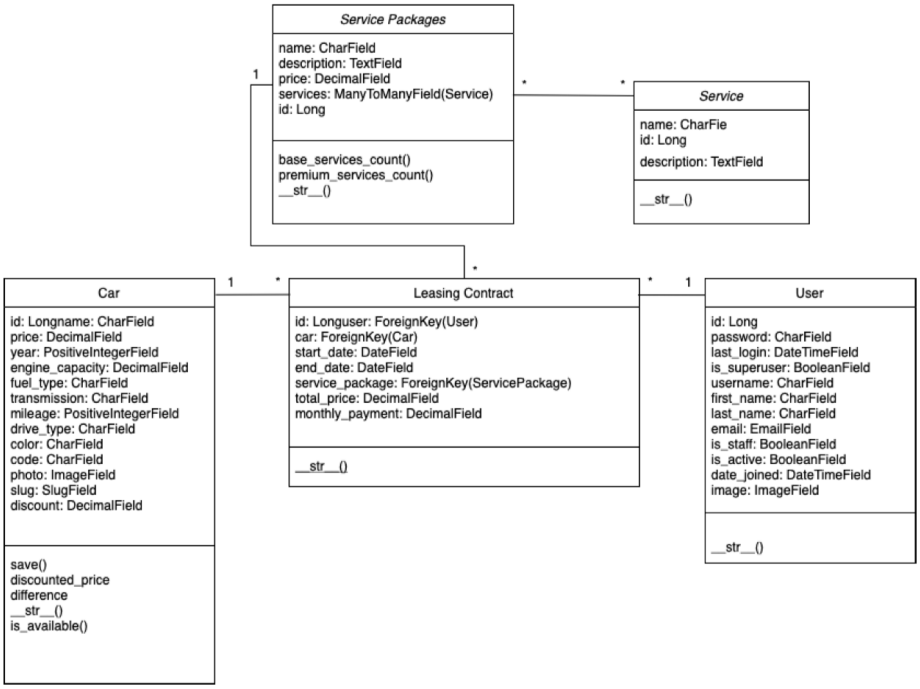


Рис. 3.4 Діаграма класів

3.3.5 Діаграма компонентів

Діаграма компонентів- це діаграма, де показуються компоненти,залежності та зв‘язки між ними.

Це схема зображення архітектури web-застосунка для надання послуг лізингу авто, яка показує, як частини системи взаємодіють між собою.

Основні частини:

1. Форми (Forms): компонент, що відповідає за взаємодію з користувачем через введення даних.

2. Маршрутизація: компонент, що керує маршрутами в застосунку, перенаправляючи запити користувачів.

3. Контролери :компонент, який обробляє запити користувачів.

4. Моделі: компонент, що презентує логіку та структуру даних, оброблює взаємодії з базами даних.

5. Представлення: компонент, що презентує дані користувача, формує інтерфейс користувача.

Головні сутності системи:

1. Користувачі: база даних для зберігання персональної інформації про користувачів.

2. Автомобілі: база даних для зберігання інформації про автомобілі в наявності.

3. Лізингові контракти: база даних для зберігання інформації про персональні угоди користувачів.

Ця структура дозволяє чітко поділити обов'язки між різними частинами web-застосунка, що робить її розробку та підтримку зручнішою та ефективною.

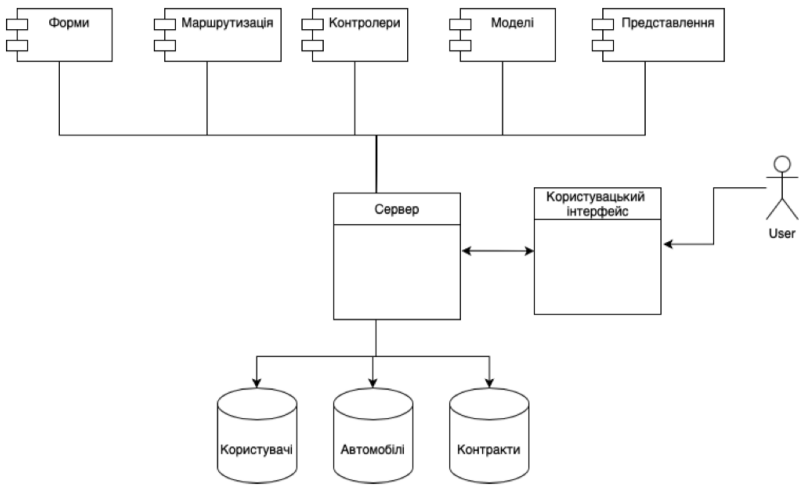


Рис. 3.5 Діаграма компонентів

4. ОПИС РОЗРОБКИ ТА ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ

4.1 Структура проєкту

Структура демонструє папки та компоненти Django-проекту для надання послуг лізингу автомобілів. Вона містить основні папки, які мають своє призначення:

1. Cars: включає в себе код пов’язаний з авто.
2. Chat: включає в себе код пов’язаний з AI-чатом, який допомагає користувачам вирішувати їх персональні питання.
3. djangoproject: коренева папка проєкту.
4. venv: оточення Python для встановлення бібліотек та залежностей проєкту.
5. leasing: процес надання послуг лізингу для користувачів.
6. main: головна папка проєкту.
7. media: фото та медіа web-застосунку.
8. none_modules: бібліотеки для роботи з фронтенд частиною.
9. services: послуги про типи лізингу.
10. templmates: HTML-шаблони.
11. users: управління користувачами.

Ця структура допомагає зручно розділяти функціонал від бекенду до фронтенд частини.

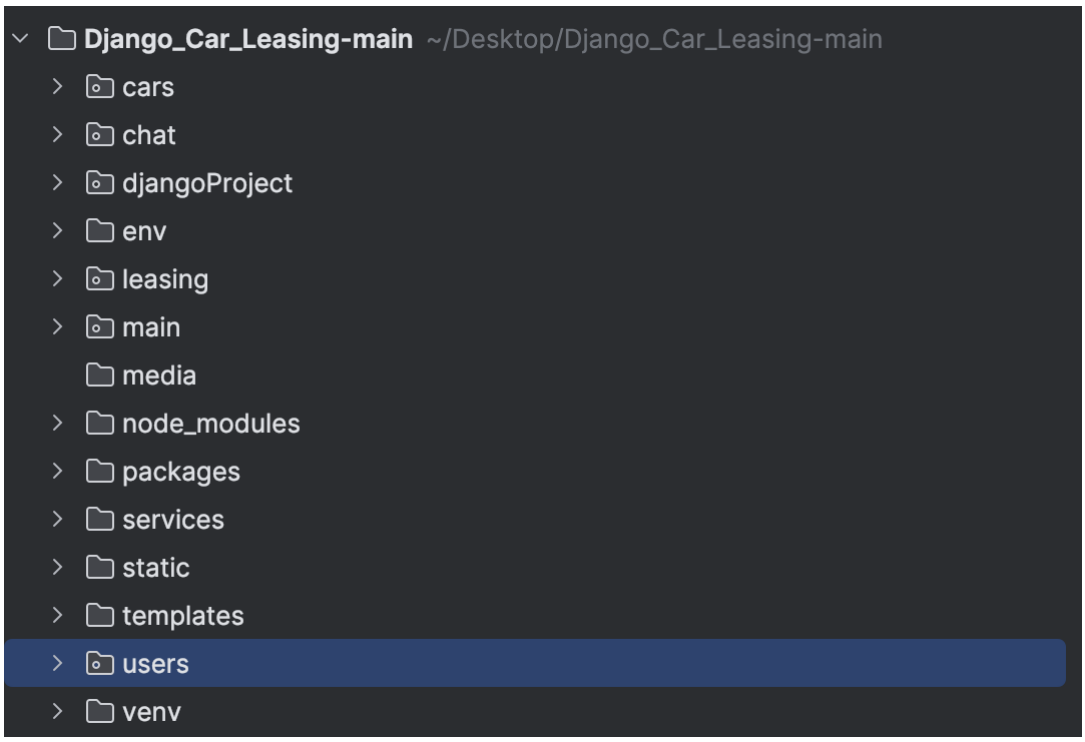


Рис. 4.1 Структура проєкту

4.2 Процес розробки реєстрація та авторизації користувачів

Код реалізує повноцінну систему аутентифікації користувачів у web-застосунку для надання послуг лізингу авто, що забезпечує реєстрацію нових користувачів, вхід у систему, управління профілем та вихід.

Функція авторизації login обробляє запити на вхід, перевіряє введені дані за допомогою форми UserLoginForm та виконує аутентифікацію через вбудований механізм Django. При коректному вході система демонструє повідомлення з ім'ям користувача та перенаправляє на головну сторінку або зазначену через параметр 'next' URL-адресу. Якщо виникають якісь помилки валідації форма відображається з відповідними повідомленнями.

```python
def login(request):
    if request.method == 'POST':
        form = UserLoginForm(data=request.POST)
        if form.is_valid():
            username = request.POST['username']
            password = request.POST['password']
            user = auth.authenticate(username=username, password=password)
            if user:
                auth.login(request, user)
                messages.success(request, message: f"{username}, Ви увійшли у акаунт")

                if request.POST.get('next', None):
                    return HttpResponseRedirect(request.POST.get('next'))

                return HttpResponseRedirect(reverse('main:index'))
    else:
        form = UserLoginForm()

    context = {
        'title': 'Home - Авторизація',
        'form': form
    }
    return render(request, template_name: 'users/login.html', context)
```

Рис. 4.2 Функція авторизації login

Реєстрація нових користувачів відбувається через функцію registration, яка використовує форму UserRegistrationForm Після вдалого заповнення даних система автоматично створює кабінет для користувача, виконує його вхід у систему та повідомляє про успішну реєстрацію, перенаправляючи на головну сторінку. Це забезпечує комфортний досвід без необхідності додаткового входу після реєстрації.

```python
def registration(request):
    if request.method == 'POST':
        form = UserRegistrationForm(data=request.POST)
        if form.is_valid():
            form.save()
            user = form.instance
            auth.login(request, user)
            messages.success(request, message: f"{user.username}, Ви успішно створили аккаунт")
            return HttpResponseRedirect(reverse('main:index'))
    else:
        form = UserRegistrationForm()

    context = {
        'title': 'Home - Регістрація',
        'form': form
    }
    return render(request, template_name: 'users/registration.html', context)
```

Рис. 4.3 Функція registration

Функція logout забезпечує швидкий та безпечний вихід з системи, після чого користувач отримує повідомлення та переходить на сторінку каталогу.

```python
@login_required
def logout(request):
    messages.success(request, message: f"{request.user.username}, Ви вийшли з аккаунту")
    auth.logout(request)
    return redirect(reverse('catalog:index'))
```

Рис. 4.4 Функція logout

4.3 Процес розробки створення особистого контракту та перегляду щомісячних платежів.

Функція profile, яка надає доступ до особистого кабінету. Вона допомагає користувачам змінювати свої дані через форму ProfileForm, а також відображає інформацію про орендні контракти. Тут реалізована систему, яка автоматично рахує щомісячні платежі на основі тривалості контракту та його загальної вартості.Всі фінансові питання виконуються з високою точністю за допомогою Decimal, а робота з датами враховує часові зони через timezone.

```python
    for contract in contracts:
        end_date = contract.end_date
        start_date = contract.start_date
        days_difference = (end_date - start_date).days
        monthly_payment = contract.total_price / Decimal(days_difference / 30)
        if start_date > today_date:
            payment_info = {
                'contract': contract,
                'message': f"Перша оплата: {start_date} - {round(monthly_payment, 2)} $"
            }
        else:
            payment_info = {
                'contract': contract,
                'message': f"Сума до сплати за цей місяць: {round(monthly_payment, 2)} $"
            }
        monthly_payments.append(payment_info)
        contract.monthly_payment = round(monthly_payment, 2)

    context = {
        'user': user,
        'contracts': contracts,
        'monthly_payments': monthly_payments,
        'today_date': today_date,
        'form': form
    }

    return render(request, template_name: 'users/profile.html', context)
```

Рис. 4.5 Функція profile

4.4 Процес розробки оформлення лізингу

Процес оформлення лізингових контрактів реалізована через функцію create contract, яка забезпечує повний цикл створення нового договору лізингу для конкретного автомобіля.

Користувач обирає авто після цього система починає роботу з отримання даних про конкретний автомобіль за його унікальним кодом. Для авторизованих користувачів (обов'язкова умова через декоратор @login_required) наступним кроком є завантаження всіх доступних сервісних пакетів та автоматичний розрахунок вартості.При заповненні форми створення контракту система реалізує багаторівневу перевірку введених даних. Спочатку стандартна валідація форми Django, потім низка додаткових перевірок: чи обрана дата початку в майбутньому, чи коректно вказаний період дії (мінімум 30 днів), чи не перетинається обраний період з існуючими контрактами на цей автомобіль. Кожна помилка супроводжується зрозумілим повідомленням для користувача. Якщо всі перевірки пройдені успішно, система виконує точні розрахунки.

Після збереження контракту система автоматично прив'язує його до користувача та обраного автомобіля, показує повідомлення про успішне оформлення та перенаправляє в персональний кабінет.

```
def create_contract(request, car_code):
    car = get_object_or_404(Car, code=car_code)
    service_packages = ServicePackage.objects.all()
    selected_package = None
    error_message = None
    car_part = (car.price / 100) * 2
    calculated_prices = []

    for package in service_packages:
        total_price = package.price * 12 + car_part
        calculated_prices.append({
            'package': package,
            'total_price': round(total_price, 2),
            'monthly_payment': round(total_price / 12, 2),
        })

    if request.method == 'POST':
        form = LeasingContractForm(request.POST)
        if form.is_valid():
            contract = form.save(commit=False)
            contract.user = request.user
            contract.car = car
```

Рис. 4.6 Функція create_contract

```
if request.method == 'POST':
    form = LeasingContractForm(request.POST)
    if form.is_valid():
        contract = form.save(commit=False)
        contract.user = request.user
        contract.car = car

        # Додаткові перевірки дат
        today = date.today()
        if contract.start_date <= today:
            error_message = _("Дата початку повинна бути у майбутньому.")
        elif contract.end_date <= contract.start_date:
            error_message = _("Дата завершення повинна бути після дати початку.")
        elif (contract.end_date - contract.start_date).days < 30:
            error_message = _("Мінімальний термін лізингу - 30 днів.")
        else:
            # Перевірка на перетин дат
            overlapping = LeasingContract.objects.filter(
                car=car
            ).filter(
                Q(start_date__lte=contract.end_date) &
                Q(end_date__gte=contract.start_date)
            ).exclude(user=request.user)
```

Рис. 4.7 Процес створення особистого контракту

```
    if overlapping.exists():
        error_message = _("Цей автомобіль вже заброньований на обраний період.")
    else:

        months = ((contract.end_date.year - contract.start_date.year) * 12 +
                  (contract.end_date.month - contract.start_date.month))
        if contract.end_date.day < contract.start_date.day:
            months -= 1
        months = max(months, 1)  # Мінімум 1 місяць

        contract.total_price = contract.service_package.price * months + car_part
        contract.monthly_payment = contract.total_price / months
        contract.save()
        messages.success(request, _("Контракт успішно створено!"))
        return redirect(reverse('users:profile'))
```

Рис. 4.8 Процес створення особистого контракту

4.5 Процес розробки AI-помічника для вирішення питань користувачів

AI-чат, допомагає надавати персоналізовані відповіді на запитання користувачів з використанням штучного інтелекту від Google - моделей Gemini. Основа роботи чату полягає в інтеграції з API Gemini через офіційну бібліотеку google.generativeai, що забезпечує доступ до передових можливостей обробки природної мови.

```
import google.generativeai as genai
```

Рис. 4.8 Бібліотека google.generativea

Ключовим елементом системи є вибір API-ключа для Gemini. Для роботи з API необхідно отримати унікальний-власний ключ у Google AI Studio, який потім зберігається у налаштуваннях Django. Вибір саме Gemini API обґрунтований кількома факторами: якісна генерація відповідей, підтримка української мови, стабільна робота та наявність безкоштовного тарифу з лімітами, достатніми для більшості персональних застосунків.

Основним аспектом є функція chat_index, яка обробляє запити користувачів через веб-інтерфейс. При GET-запиті вона відображає сторінку чату, а при POST-запиті - робить аналіз повідомлення користувача та генерує відповідь за допомогою обраної моделі Gemini. Система підтримує два варіанти моделей: gemini-1.5-flash-latest для швидкої обробки простих запитів та gemini-pro для складніших питань, що вимагають детального аналізу.

```
def chat_index(request):
    if request.method == 'GET':
        return render(request, template_name: 'chat/index.html')

    if not gemini_model and not initialize_gemini():
        return JsonResponse( data: {
            'error': 'AI сервіс недоступний. Спробуйте пізніше.',
            'status': 'service_unavailable'
        }, status=503)

    try:

        if request.content_type == 'application/json':
            data = json.loads(request.body)
        else:
            data = request.POST

        user_message = data.get('message', '').strip()
        if not user_message:
            return JsonResponse( data: {
                'error': 'Порожнє повідомлення',
                'status': 'bad_request'
            }, status=400)
```

Рис. 4.9 Функція chat_index

4.6 Клієнтська частина web-застосунку

Інтерфейс розробленого web-застосунку складається з наступних сторінок та елементів:

1. Головна сторінка: за допомогою цієї сторінки можна здійснювати перехід між іншими сторінками та побачити авто в наявності.

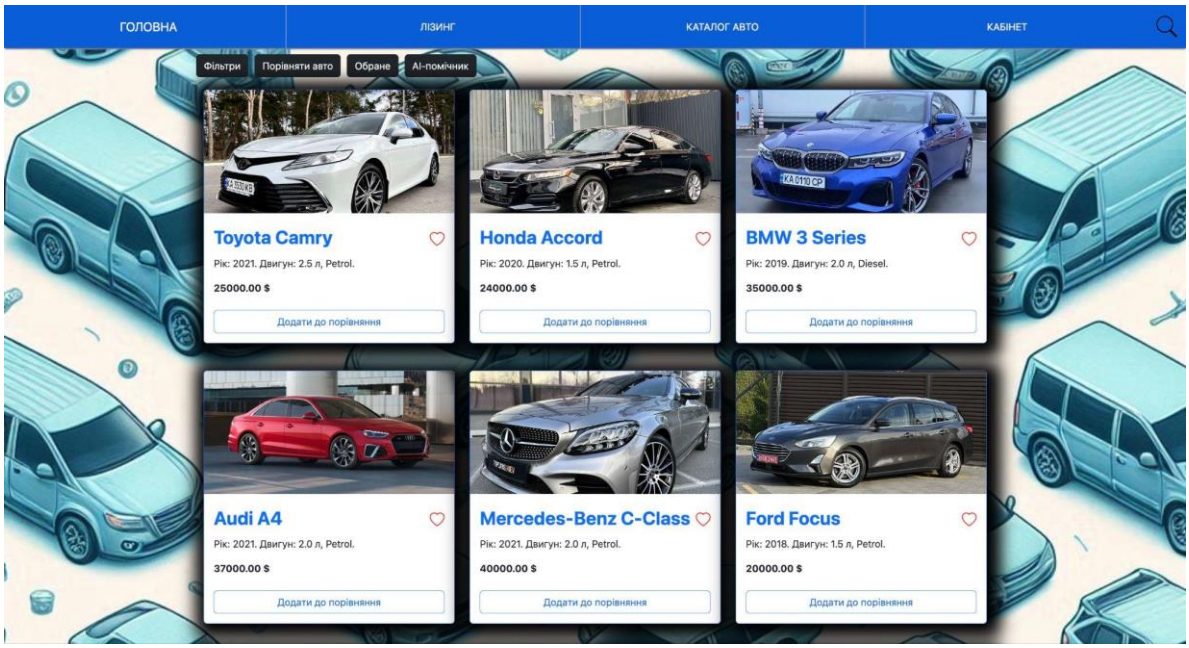


Рис. 4.10 Дизайн головної сторінки

2. Сторінка про детальну інформацію авто: користувачі можуть переглядати детальну інформацію про обраний автомобіль.

1. Натисніть на автомобіль у каталозі.

2. Користувач може побачити деталі, такі як технічні характеристики, умови лізингу та фото:

1. Характеристика авто.

2. Ціна авто.

3. Кнопку “Оформлення лізингу”

4. Перегляд інформації про тип лізингу

5.Натисніть на автомобіль у каталозі.

6.Користувач може побачити деталі, такі як технічні характеристики, умови лізингу та фото.

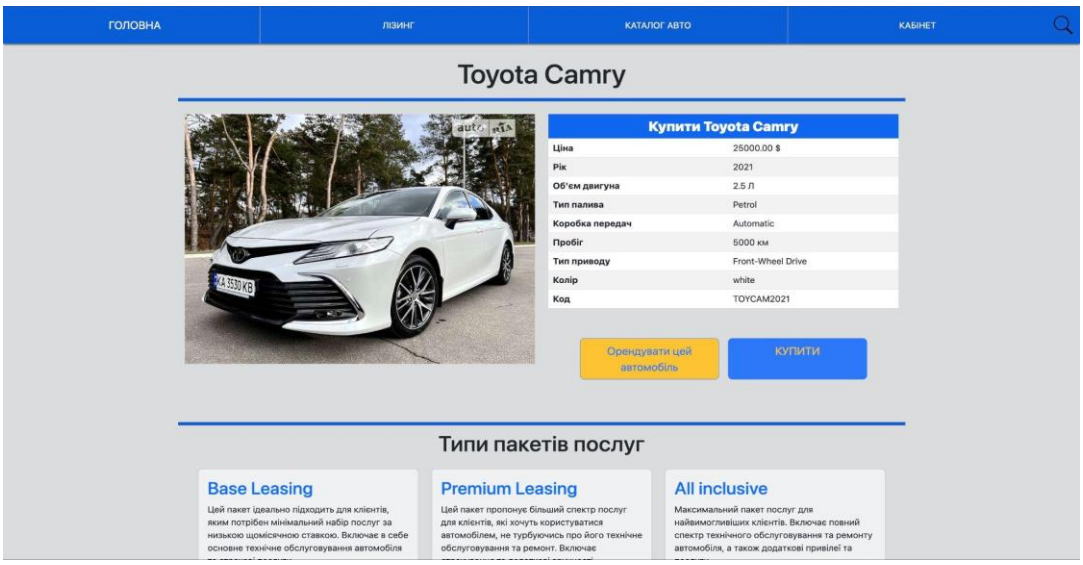


Рис. 4.10 Дизайн сторінки авто

3. Сторінка оформлення лізингу: користувачі можуть заповнити форму для оформлення договору лізингу на автомобіль.

1. На сторінці інформації про автомобіль натисніть кнопку "Взяти в лізинг".

2. Заповніть необхідні дані у формі.

3. Натисніть кнопку "Відправити".

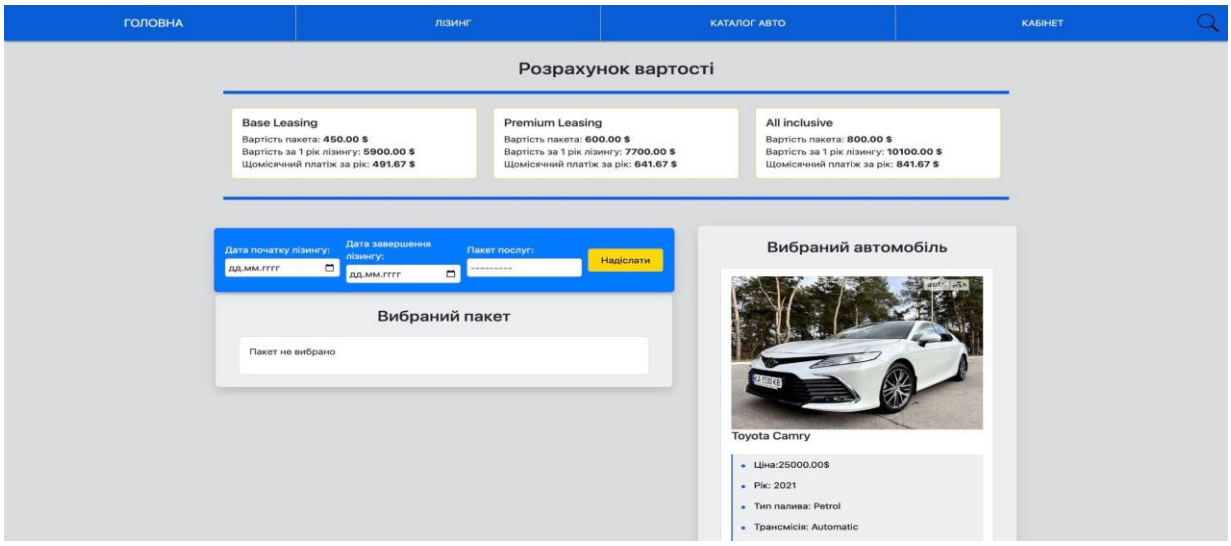


Рис. 4.11 Дизайн сторінки оформлення лізингу

4. Сторінки реєстрації та авторизації: користувач повинен написати своє ім’я, прізвище, ім’я користувача, e-mail та придумати гарний пароль, після цих дій

буде створено персональний кабінет. Для входу в кабінет потрібно вести правильне ім’я користувача та пароль.

1. Перейдіть на сторінку реєстрації.

2. Заповніть форму реєстрації, вказавши електронну пошту, ім'я користувача та пароль.

3. Натисніть кнопку "Зареєструватися".

Рис. 4.12 Дизайн сторінки реєстрації та авторизації

5. Сторінки перегляду інформації про лізинг та типи лізингу: сторінка перегляду інформації про лізинг містить відповіді на самі популярні питання про лізинг. Сторінка типи лізингу містить інформацію про типи лізингу, які надає компанія.

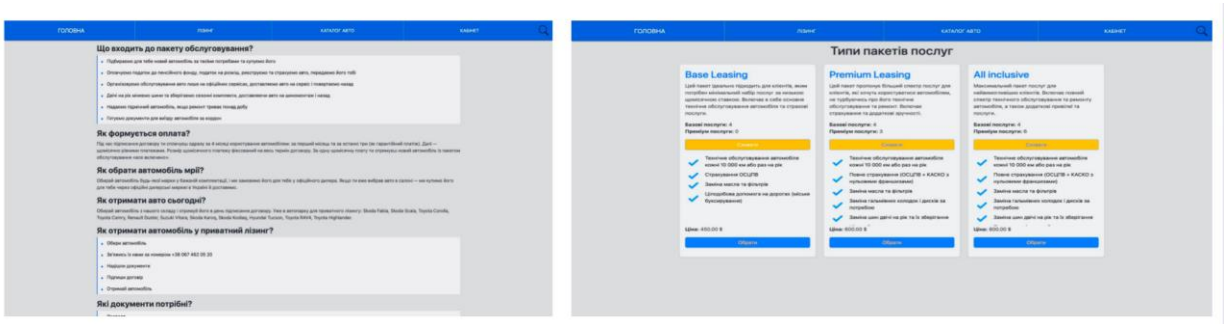


Рис. 4.13 Дизайн сторінки перегляду інформації про лізинг та типи лізингу

6. Сторінка персонального кабінету:ця сторінка містить про дані користувача та перегляд інформації про лізинг та персонального контракту.

1.Користувач повинен увійти до свого персонального кабінету.

2. Перейдіть до розділу "Кабінет користувача".

3. Перегляньте список укладених контрактів та їх детальну інформацію.

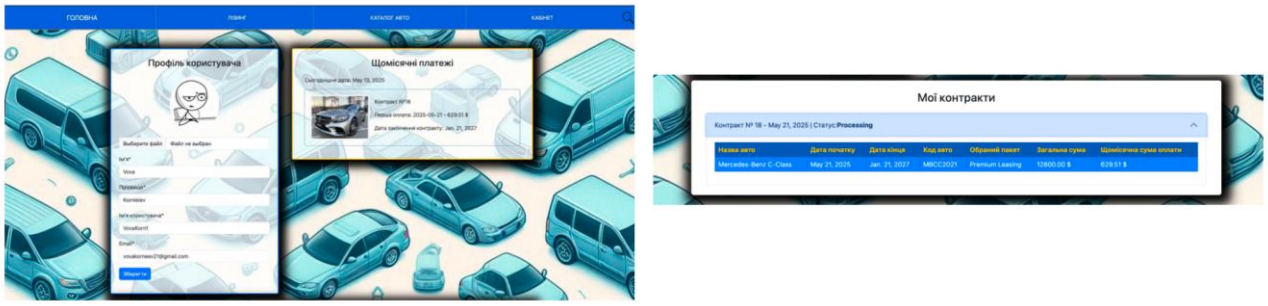


Рис. 4.14 Дизайн сторінки персонального кабінету

4.7 Тестування web-застосунку

Мануальне тестування — це тестування програмного забезпечення, яке проводиться для знаходження помилок вручну[15].

Було проведено мануальне тестування web-застосунку, опис відповідних тестів наведено у таблицях.

Таблиця 3.1

Тест 1.1

Умови	Результати
Тест	Вхід користувача в персональний кабінет.
Модуль	Вхід у профіль.
Номер тесту	1.1
Початковийстан системи	Користувач знаходиться на сторінці входу у особистий кабінет.
Вхідні дані	Ім’я користувача, пароль.
Опис проведення тесту	У відповідні поля вводяться: правильний пароль та логін, які співпадають з існуючим записом у БД. Після цього натискається кнопка відправки паролю.
Очікуваний результат	Вхід проходить успішно, користувач входить в персональний кабінет і переходить на головну сторінку web-застосунку.
Фактичний результат	Вхід проходить успішно, користувач входить в персональний кабінет і переходить на головну сторінку web-застосунку.

Таблиця 3.2

Тест 1.2

Умови	Результати
Тест	Неправильний вхід користувача.
Модуль	Вхід у профіль.
Номер тесту	1.2
Початковий стан системи	Користувач знаходиться на сторінці входу в персональний кабінет.
Вхідні дані	Логін, невірний пароль
Опис проведення тесту	У відповідні поля вводяться: коректний логін та невірний пароль. Після цього натискається кнопка відправки даних.
Очікуваний результат	Вхід не проходить, система показує повідомлення про помилку.
Фактичний результат	Вхід не проходить, система показує повідомлення про помилку.

Таблиця 3.3

Тест 2.1

Умови	Результати
Тест	Реєстрація нового користувача.
Модуль	Реєстрація.
Номер тесту	2.1
Початковий стан системи	Користувач знаходиться на сторінці реєстрації.

Продовження таблиці 3.3

Тест 2.1

Умови	Результати
Вхідні дані	Логін, пароль, підтвердження паролю, електронна пошта , ім’я, прізвище.
Опис проведення тесту	У поля вводяться дані для реєстрації: логін, пароль, підтвердження паролю, електронна пошта, ім’я, прізвище. Після цього натискається кнопка реєстрації.
Очікуваний результат	Реєстрація проходить швидко та успішно, користувач створюється у системі і переходить на сторінку входу.
Фактичний результат	Реєстрація проходить швидко та успішно, користувач створюється у системі і переходить на сторінку входу.

Таблиця 3.4

Тест 3.1

Умови	Результати
Тест	Перегляд каталогу автомобілів.
Модуль	Каталог автомобілів.
Номер тесту	3.1
Початковий стан системи	Користувач знаходиться на сторінці каталогу авто.
Вхідні дані	Вибір фільтрів пошуку автомобілів.
Опис проведення тесту	Користувач переходить до сторінки каталогу автомобілів, обирає фільтри та натискає кнопку пошуку.
Очікуваний результат	Система показує список автомобілів, що відповідають обраним фільтрам.
Фактичний результат	Система показує список автомобілів, що відповідають вибраним фільтрам.

Таблиця 3.5

Тест 3.2

Умови	Результати
Тест	Порівняння автомобілів.
Модуль	Каталог автомобілів.
Номер тесту	3.2
Початковий стан системи	Користувач знаходиться на сторінці каталогу авто.
Вхідні дані	Вибір обраних автомобілів для порівняння.

Продовження таблиці 3.5

Тест 3.2

Умови	Результати
Опис проведення тесту	Користувач переходить до сторінки каталогу автомобілів, обирає обрані авто та натискає кнопку порівняння.
Очікуваний результат	Система показує таблицю порівняння характеристик автомобілів,які обрав користувач.
Фактичний результат	Система показує таблицю порівняння характеристик автомобілів,які обрав користувач.

Таблиця 3.6

Тест 3.3

Умови	Результати
Тест	Додавання авто в обране.
Модуль	Каталог автомобілів.
Номер тесту	3.3
Початковий стан системи	Користувач знаходиться на сторінці каталогу авто.
Вхідні дані	Вибір обраних автомобілів.
Опис проведення тесту	Користувач переходить до сторінки каталогу автомобілів, обирає обрані авто та натискає кнопку у вигляді сердечка.
Очікуваний результат	Система показує додані авто в обране.
Фактичний результат	Система показує додані авто в обране.

Таблиця 3.7

Тест 3.4

Умови	Результати
Тест	Відповідь AI-чату на запитання користувача.
Модуль	Каталог автомобілів.
Номер тесту	3.4
Початковий стан системи	Користувач відкрив вікно чату з AI-помічником.
Вхідні дані	Користувач пише питання,яке його цікавить.
Опис проведення тесту	Користувач вводить питання питання та натискає кнопку “Надіслати”.

Продовження таблиці 3.7

Тест 3.4

Умови	Результати
Очікуваний результат	Система надає точну відповідь на питання користувача.
Фактичний результат	Система надає точну відповідь на питання користувача.

Таблиця 3.8

Тест 4.1

Тест	Оформлення лізингу.
Модуль	Договір лізингу
Номер тесту	4.1
Початковий стан системи	Користувач авторизований, переглядає інформацію обраного автомобіля
Вхідні дані	Персональні дані, умови лізингу
Опис проведення тесту	Користувач натискає кнопку "Взяти в лізинг".
Очікуваний результат	Успішно створений договір, користувач отримує повідомлення про успішне оформлення лізингу.
Фактичний результат	Успішно створений договір, користувач отримує повідомлення про успішне оформлення лізингу.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено web-застосунок для надання послуг лізингу авто. Актуальність цієї роботи полягає в тому, що зараз web-застосунок є одним ефективних способів ведення бізнесу, який в свою чергу відіграє важливу роль в економіці.

1. Проведено аналіз предметної галузі, відповідно до теми роботи визначено переваги web-застосунку, як спосіб ведення бізнесу – надання послуг лізингу авто. Проаналізовано три існуючих web-застосунки для лізингу авто: ХапайАвто, Vidi, AutoRia, показано їх основні переваги та недоліки. В основі аналізу предметної галузі та аналогів, визначено технічне завдання та розроблено функціональні та нефункціональні вимоги до web-застосунку.

Ключевими з них є:

-Реєстрація та авторизація.

-Пошук автомобіля за характеристиками.

-Вибір фільтрів підбору авто по ціні.

-Додавання авто в улюблене.

-Порівняння авто за характеристиками.

-Перегляд інформації про типи та види лізингу.

-Вибір пакету лізингу для автомобіля.

-Перегляд діючих контрактів та діючих оплат (загальна сума та щомісячна оплата).

- AI-помічник

-Збереження даних користувача в захищеному вигляді.

-Забезпечення точності пошуку автомобіля.

-Зручний інтерфейс для фільтрації та порівняння авто.

-Сумісність з сучасними браузерами (Safari, Google Chrome, Opera)

-Швидке завантаження сторінок сайту (не більше 3 секунд)

2. Проаналізовано існуючі засоби розробки web-застосунків та обрано наступні: фреймворк Django, база даних PostgreSQL, середовища розробки PyCharm, Git та GitHub для контролю версій.

3. Розроблено web-застосунок для надання послуг лізингу авто.

4. Проведено мануальне тестування web-застосунка та перевірено виконання функціональних та нефункціональних вимог.

5. Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

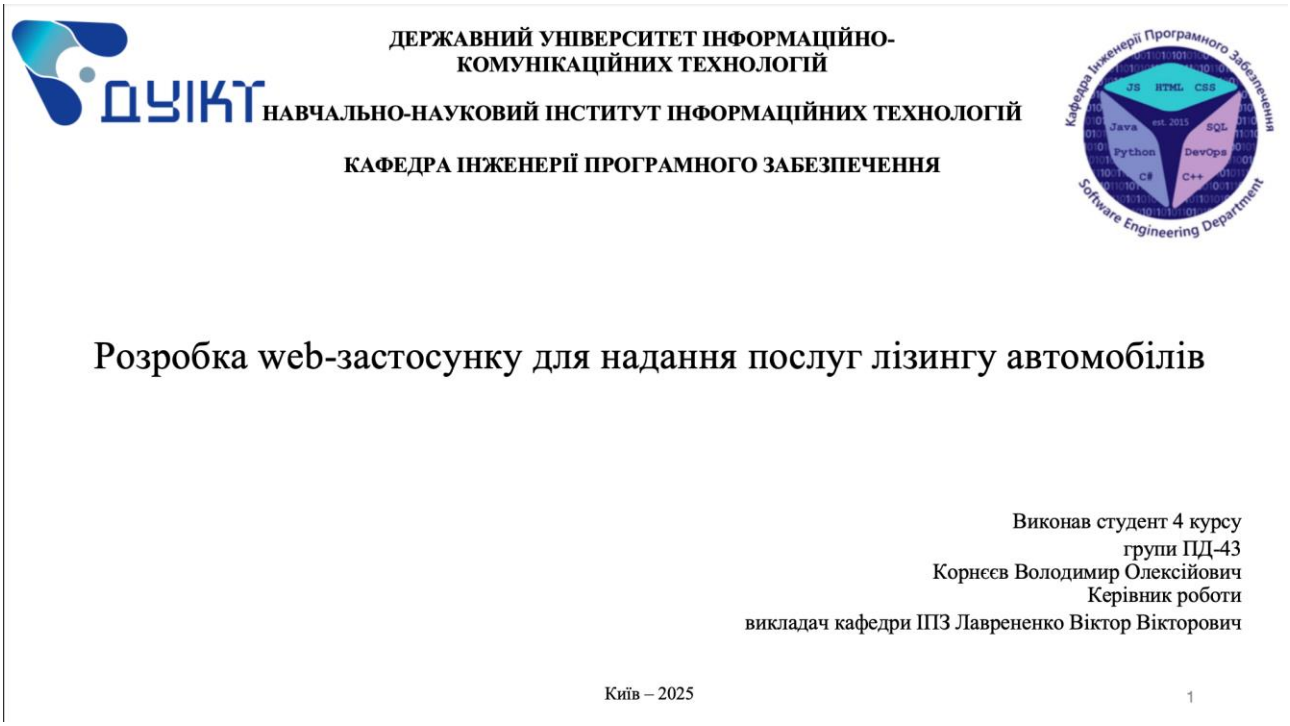
1. Корнєєв В.О., Лаврененко В.В. Огляд сучасних технологій для розробки web-застосунку для надання послуг лізингу автомобілів// VI Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. 15 квітня 2025 року, ДУІКТ, Київ (подано до друку).

2. Корнєєв В.О., Лаврененко В.В. Забезпечення конфіденційності інформації у web-застосунку для надання послуг лізингу автомобілів // Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна С. 287-288.

ПЕРЕЛІК ПОСИЛАНЬ

1. Автолізинг у 2025 році: що змінилось і чому це зручно. Protocol. URL: https://protocol.ua/ua/avtolizing_u_2025_rotsi_shcho_zminilos_i_chomu_tse_zruchno/
2. Що таке лізинг і що означає взяти в лізинг авто. ULF. URL: https://ulf.ua/post/chto-takoe-lizing-i-chto-oznachaet-vzyat-v-lizing-avto
3. 10 кращих ідей для розробників: вибір професіоналів. Softlist. URL: https://softlist.com.ua/ua/news/desiat-krashchih-ide-dlya-rozrobnikiv-vibir-profesionaliv
4. PyCharm: як встановити та використовувати для Python. GoIT. URL: https://goit.global/ua/articles/pycharm-iak-vstanovyty-ta-vykorystovuvaty-dlia-python/
5. PyCharm — це...? Foxminded. URL: https://foxminded.ua/pycharm-tse/
6. Вступ до Python. Acode. URL: https://acode.com.ua/intro-python/
7. Що таке JavaScript і для чого він потрібен. GoIT. URL: https://goit.global/ua/articles/shcho-take-javascript-i-dlia-choho-vin-potriben/
8. HTML і CSS: що це, кому та для чого потрібно. GoIT. URL: https://goit.global/ua/articles/html-i-css-shcho-tse-komu-ta-dlia-choho-potribno/
9. Django Girls Tutorial. URL: https://tutorial.djangogirls.org/uk/django/
10. Встановлення PostgreSQL. TheHost. URL: https://thehost.ua/ua/wiki/administration/database/postgresql-install
11. Що таке GitHub і як з ним працювати. QATestLab. URL: https://training.qatestlab.com/blog/technical-articles/what-is-github-and-how-to-work/
12. Що таке діаграма варіантів використання UML. MindOnMap. URL: https://www.mindonmap.com/uk/blog/what-is-a-uml-use-case-diagram/
13. Діаграма активностей UML. Guru99. URL: https://www.guru99.com/uk/uml-activity-diagram.html
14. Діаграма класів UML. MindOnMap. URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/#part1
15. Що таке мануальне тестування?. DAN.IT. URL: https://dan-it.com.ua/uk/blog/v-chem-raznica-mezhdu-avtomatizirovannym-i-manualnym-qa/

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для надання послуг лізингу автомобілів

Виконав студент 4 курсу
групи ПД-43
Корнєєв Володимир Олексійович
Керівник роботи
викладач кафедри ІПЗ Лаврененко Віктор Вікторович

Київ – 2025

1

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: допомога у поширенні послуг , що забезпечить швидкий і зручний доступ до послуг лізингу автомобілів.

Об’єкт дослідження: процес надання послуг лізингу автомобілів та супровід лізингових контрактів.

Предмет дослідження: web-застосунок, що забезпечує організацію та реалізацію послуг з лізингу автомобілів.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз сучасних вимог до веб-застосунків.
2. Здійснити огляд та аналіз існуючих платформ для надання послуг лізингу автомобілів, визначити їх переваги та недоліки.
3. Провести огляд засобів для розробки програмного забезпечення клієнт-серверного застосунку для надання послуг лізингу автомобілів.
4. Сформувати вимоги до розробки клієнт-серверного застосунку.
5. Спроєктувати архітектуру клієнт-серверного застосунку .
6. Розробити та протестувати клієнт-серверний застосунок для надання послуг лізингу автомобілів.

АНАЛІЗ АНАЛОГІВ

Показник	ХапайАвто	Vidi	AvtoRia	DreamCarLeasing
Особистий кабінет	+	+	+	+
Оформлення лізингу	+	+	+	+
Порівняння авто	-	-	+	+
Фільтрація за ціною	+	+	+	+
Додавання авто у улюблене	-	+	+	+
Пошук авто	-	-	+	+
AI-асистент	-	-	-	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Реєстрація та авторизація.
2. Пошук автомобіля за характеристиками.
3. Вибір фільтрів підбору авто по ціні.
4. Додавання авто в улюблене.
5. Порівняння авто за характеристиками.
6. Перегляд інформації про типи та види лізингу.
7. Вибір пакету лізингу для автомобіля.
8. Перегляд діючих контрактів та діючих оплат (загальна сума та щомісячна оплата).

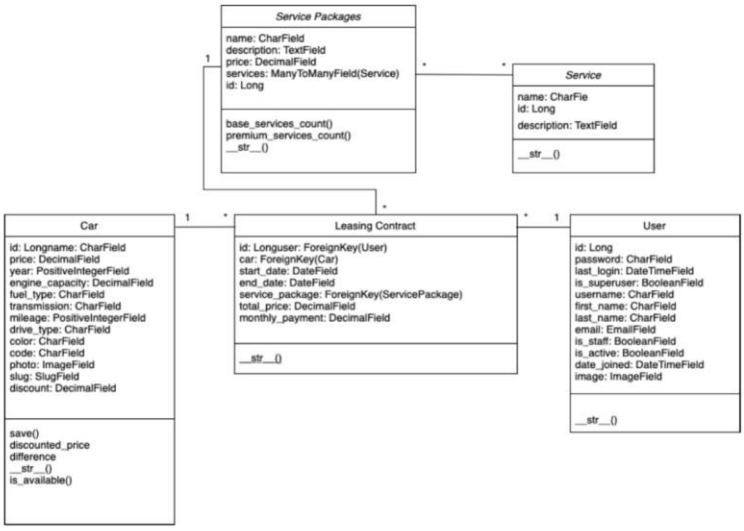
Нефункціональні вимоги:

1. Збереження даних користувача в захищеному вигляді.
2. Забезпечення точності пошуку автомобіля.
3. Зручний інтерфейс для фільтрації та порівняння авто.
4. Сумісність з сучасними браузерами (Safari, Google Chrome, Opera)
5. Швидке завантаження сторінок сайту (не більше 3 секунд)

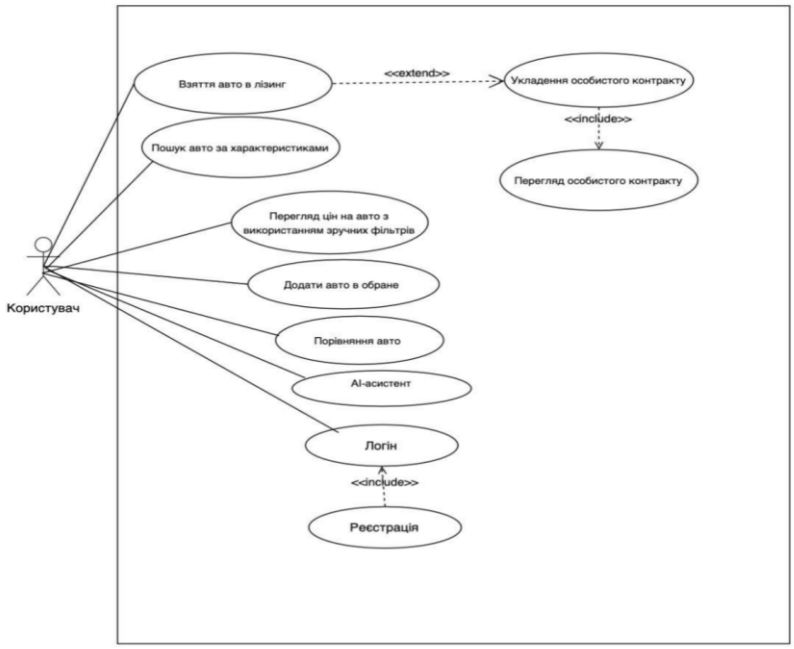
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

МАПА WEB-ЗАСТОСУНКУ

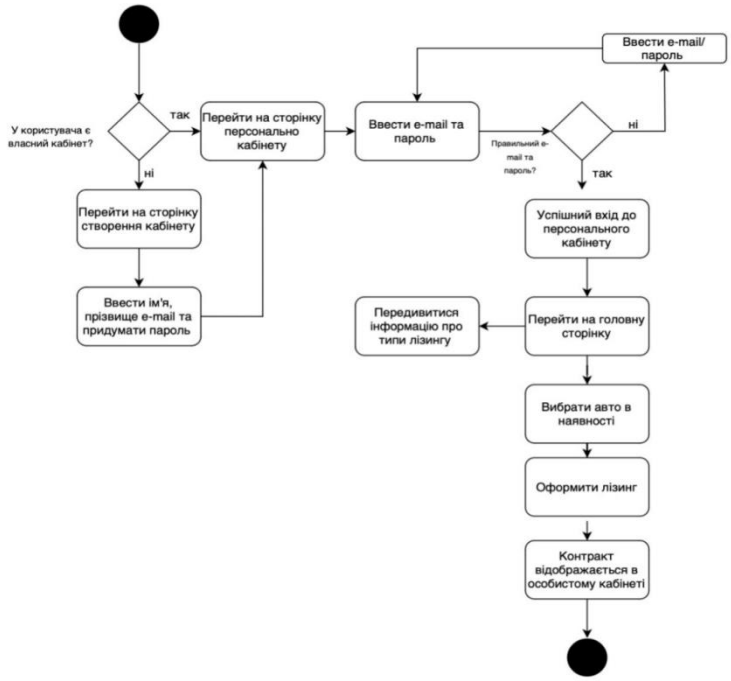
ДІАГРАМА КЛАСІВ



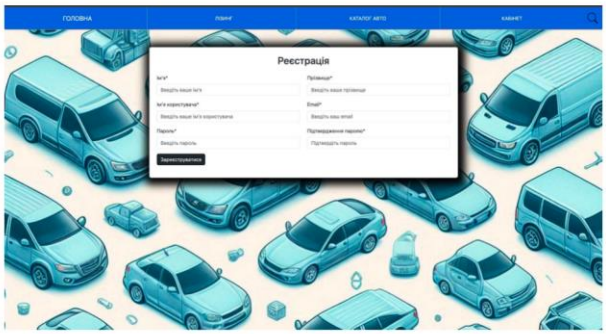
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



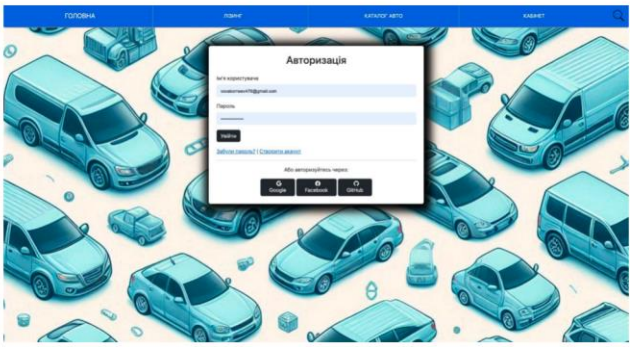
ДІАГРАМА ДІЯЛЬНОСТІ (ОФОРМЛЕННЯ ЛІЗИНГУ)



ЕКРАННІ ФОРМИ

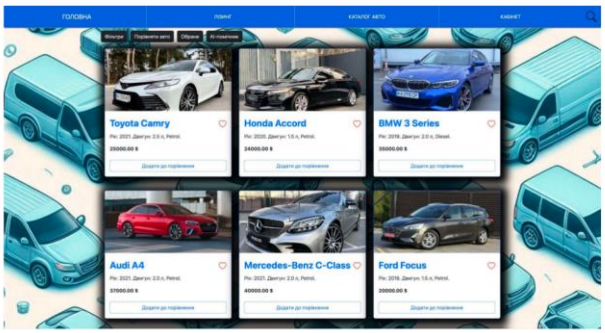


Екран реєстрації

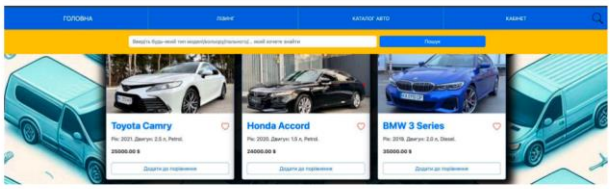


Екран авторизації

ЕКРАННІ ФОРМИ

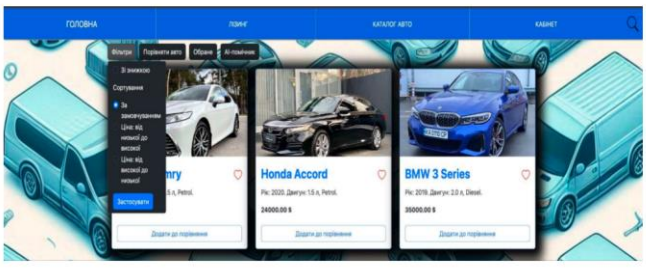


Екран список авто

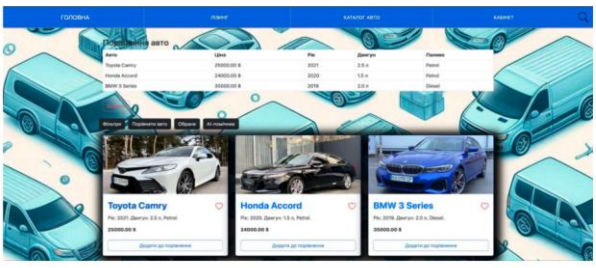


Екран пошук авто за характеристиками

ЕКРАННІ ФОРМИ

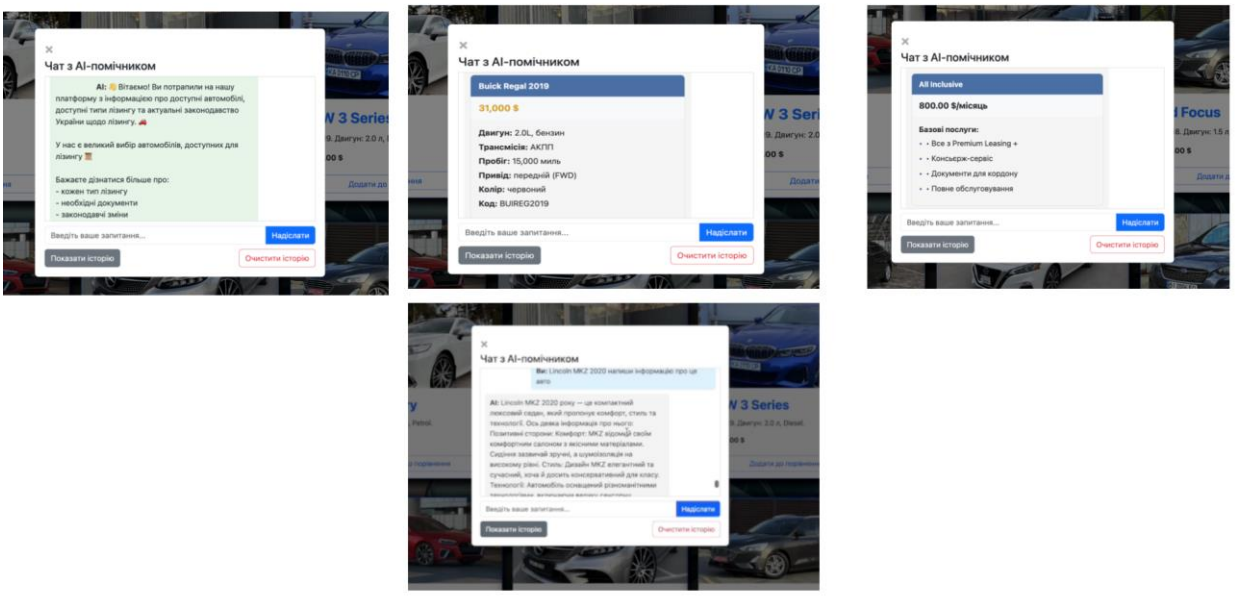


Екран фільтр ціни



Екран порівняння авто

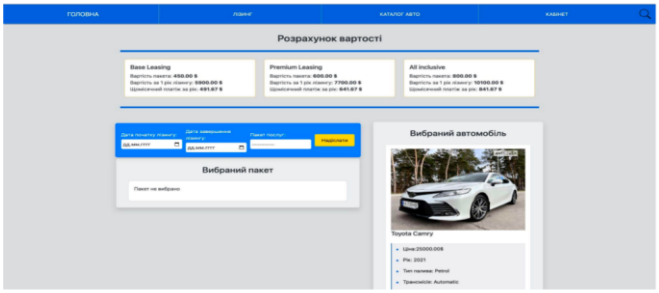
ЕКРАННІ ФОРМИ



Екран з AI-асистентом

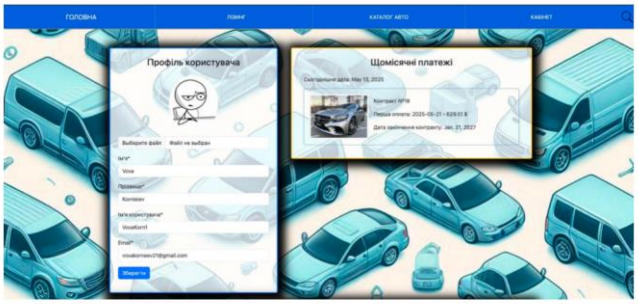
ЕКРАННІ ФОРМИ

Екран оформлення лізингу

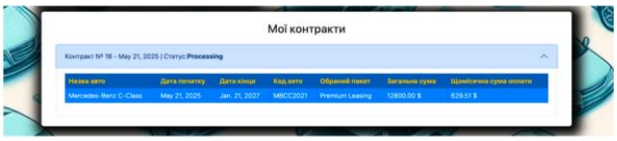


Екран оформлення лізингу

ЕКРАННІ ФОРМИ



Екран особистого кабінету



Екран перегляду особистих контрактів

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Корнєєв В.О., Лаврененко В.В. Огляд сучасних технологій для розробки web-застосунку для надання послуг лізингу автомобілів// VI Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. 15 квітня 2025 року, ДУІКТ, м.Київ (подано до друку).
2. Корнєєв В.О., Лаврененко В.В. Забезпечення конфіденційності інформації у web-застосунку для надання послуг лізингу автомобілів // Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна с. 287-288 .

ВИСНОВКИ

1. Проведено аналіз сучасних вимог до веб-застосунків, а саме: розглянуто вимоги до продуктивності, масштабованості, а також принципи UI/UX.
2. Проведено аналіз функціональності існуючих платформ для лізингу автомобілів, що дало можливість додати нову характеристику , застосування штучного інтелекту.
3. Проведено огляд технологій для розробки клієнт-серверних застосунків.
4. Сформульовано функціональні та нефункціональні вимоги вимоги до застосунку.
5. Розроблено web-застосунок на основі обраних технологій і з урахуванням зазначених вимог.
6. Проведено функціональне та нефункціональне тестування системи, яке показало, що застосунок відповідає зазначеним вимогам.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
@csrf_exempt
def gemini_chat(request):
    if request.method == 'POST':
        try:
            body = json.loads(request.body)
            user_message = body.get('message', '')

            headers = {
                "Content-Type": "application/json",
                "Authorization": f"Bearer {GEMINI_API_KEY}"
            }

            # Формат згідно офіційної документації Gemini
            data = {
                "contents": [{
                    "role": "user",
                    "parts": [{"text": user_message}]
                }]
            }

            response = requests.post(GEMINI_API_URL,
headers=headers, json=data)

            print("Gemini API response:", response.status_code,
response.text)  # для дебагу

            if response.status_code == 200:
                try:
                    gemini_reply =
response.json()['candidates'][0]['content']['parts'][0]['text']
                    return JsonResponse({'reply': gemini_reply})
                except (KeyError, IndexError):
                    return JsonResponse({'reply': 'Помилка обробки
відповіді Gemini API'}, status=500)
            else:
                return JsonResponse({
                    'reply': f'Помилка запиту до Gemini API:
{response.status_code}',
                    'details': response.text
                }, status=response.status_code)

        except json.JSONDecodeError:
            return JsonResponse({'error': 'Невірний формат JSON
у тілі запиту'}, status=400)

        except Exception as e:
            return JsonResponse({'error': f'Виникла неочікувана
помилка: {e}'}, status=500)

    return JsonResponse({'error': 'Метод запиту має бути
POST'}, status=400)


def auto(request, car_code):
    try:
        car = Car.objects.get(code=car_code)
        service_packages = ServicePackage.objects.all()
        base_leasing = ServicePackage.objects.filter(name="Base
Leasing").first()
        base_services_count = base_leasing.services.count() if
base_leasing else 0

        for package in service_packages:
            package.base_services_count = base_services_count
            package.premium_services_count = (
                package.services.count() - base_services_count if
package.name != "Base Leasing" else 0
            )

        context = {
            'car': car,
            'service_packages': service_packages
        }
        return render(request, 'auto.html', context)
    except Car.DoesNotExist:
        return JsonResponse({'error': 'Автомобіль не знайдено'},
status=404)


def catalog(request):
    order_by = request.GET.get('order_by', 'default')
    discount_filter = request.GET.get('discount', 'false') == 'true'
    query = request.GET.get('q', None)

    cars = Car.objects.all()

    if discount_filter:
        cars = cars.filter(discount__isnull=False, discount__gt=0)

    if order_by == 'price':
        cars = cars.order_by('price')
    elif order_by == '-price':
        cars = cars.order_by('-price')
    elif query:
        cars = q_search(query)

    paginator = Paginator(cars, 9)
    page = request.GET.get('page', 1)

    try:
        cars_page = paginator.page(page)
    except PageNotAnInteger:
        cars_page = paginator.page(1)
    except EmptyPage:
        cars_page = paginator.page(paginator.num_pages)

    context = {
        'cars': cars_page,
        'order_by': order_by,
        'discount_filter': discount_filter,
    }
    return render(request, 'catalog.html', context)


def search(request):
    query = request.GET.get('q', '')
    cars = Car.objects.filter(name__icontains=query) if query
else Car.objects.all()
    context = {
        'cars': cars,
        'query': query,
    }
    return render(request, 'catalog.html', context)
```

```
# Новий маршрут для отримання деталей пакету
def package_details(request, package_id):
    try:
        # Отримуємо деталі пакету за ID
        package = ServicePackage.objects.get(id=package_id)

        # Формуємо відповідь
        package_data = {
            'id': package.id,
            'name': package.name,
            'description': package.description,  # якщо є поле
опису
            'services': [service.name for service in
package.services.all()],  # список послуг
        }

        return JsonResponse({'package': package_data})
    except ServicePackage.DoesNotExist:
        return JsonResponse({'error': 'Пакет не знайдено'},
status=404)
from django.http import JsonResponse
from django.shortcuts import render
from django.views.decorators.http import
require_http_methods
from django.views.decorators.csrf import csrf_exempt
from django.conf import settings
import google.generativeai as genai
import json
import logging
import time
from datetime import datetime

logger = logging.getLogger(__name__)

# Налаштування квот та лімітів
FREE_TIER_LIMITS = {
    'models/gemini-1.5-flash-latest': {
        'requests_per_minute': 60,
        'requests_per_day': 1500,
        'tokens_per_minute': 64000,
        'min_delay': 1
    },
    'models/gemini-pro': {
        'requests_per_minute': 60,
        'requests_per_day': 1000,
        'tokens_per_minute': 32000
    }
}

# Глобальні змінні
gemini_model = None
current_model_name = None
last_request_time = 0
request_count_minute = 0
request_count_day = 0
last_minute_check = time.time()
last_day_check = time.time()

# Константи
REQUEST_DELAY = 30.0  # Мінімальна затримка між
запитами для безкоштовного тарифу (секунди)
RETRY_DELAY = 60  # Затримка при перевищенні квот
(секунди)
MAX_RETRIES = 2  # Максимальна кількість спроб
повторити запит

def initialize_gemini(model_name=None):
    global gemini_model, current_model_name
    try:
        api_key = getattr(settings, 'GEMINI_API_KEY', None)
        if not api_key:
            raise ValueError("GEMINI_API_KEY не знайдено в
settings.py")

        genai.configure(api_key=api_key)

        # Якщо модель не вказана, вибираємо першу доступну
        if not model_name:
            model_name = next(iter(FREE_TIER_LIMITS.keys()))

        if model_name not in FREE_TIER_LIMITS:
            raise ValueError(f"Модель {model_name} не
підтримується у безкоштовному тарифі")

        gemini_model = genai.GenerativeModel(model_name)
        current_model_name = model_name
        logger.info(f" Ініціалізовано модель: {model_name}")
        return True

    except Exception as e:
        logger.error(f" Помилка ініціалізації Gemini: {str(e)}")
        gemini_model = None
        current_model_name = None
        return False

def check_quotas():
    """Перевіряємо квоти та ліміти для поточної моделі"""
    global request_count_minute, request_count_day,
last_minute_check, last_day_check

    current_time = time.time()
    model_limits =
FREE_TIER_LIMITS.get(current_model_name, {})

    # Скидаємо лічильник хвилинних запитів
    if current_time - last_minute_check > 60:
        request_count_minute = 0
        last_minute_check = current_time

    # Скидаємо лічильник денних запитів
    if current_time - last_day_check > 86400:
        request_count_day = 0
        last_day_check = current_time

    # Перевіряємо ліміти
    if request_count_minute >=
model_limits.get('requests_per_minute', 0):
        raise Exception("Досягнуто ліміт запитів за хвилину")

    if request_count_day >=
model_limits.get('requests_per_day', 0):
        raise Exception("Досягнуто ліміт запитів за день")

def make_gemini_request(chat_session, user_message,
retry_count=0):
    """Виконати запит до Gemini API з обробкою помилок та
ретраями"""
    global last_request_time, request_count_minute,
request_count_day

    try:
        current_time = time.time()

        # Перевіряємо квоти перед запитом
        check_quotas()

        # Дотримуємось rate limit
        elapsed = current_time - last_request_time
```

```
        model_limits =
FREE_TIER_LIMITS.get(current_model_name, {})
        min_delay = model_limits.get('min_delay',
REQUEST_DELAY)

        if elapsed < min_delay:
            wait_time = min_delay - elapsed
            logger.info(f"□ Чекаємо {wait_time:.1f} сек для
дотримання rate limit")
            time.sleep(wait_time)

        # Оновлюємо лічильники
        request_count_minute += 1
        request_count_day += 1
        last_request_time = time.time()

        # Логуємо інформацію про запит
        logger.info(
            f"□ Запит до {current_model_name} (хвилинних:
{request_count_minute}/{FREE_TIER_LIMITS[current_mode
l_name]['requests_per_minute']}, денних:
{request_count_day}/{FREE_TIER_LIMITS[current_model_n
ame]['requests_per_day']})")

        # Виконуємо запит
        response = chat_session.send_message(
            user_message,
            generation_config=genai.types.GenerationConfig(
                temperature=0.7,
                top_p=0.9,
                top_k=40,
                max_output_tokens=2048,
            ),
            safety_settings={
                'HARM_CATEGORY_HARASSMENT':
'BLOCK_NONE',
                'HARM_CATEGORY_HATE_SPEECH':
'BLOCK_NONE',
                'HARM_CATEGORY_SEXUALLY_EXPLICIT':
'BLOCK_NONE',
                'HARM_CATEGORY_DANGEROUS_CONTENT':
'BLOCK_NONE',
            }
        )

        if response.text:
            return response
        raise Exception("Пуста відповідь від API")

    except Exception as e:
        logger.error(f"Помилка запиту до Gemini: {str(e)}")

        # Якщо це помилка квоти (429), чекаємо і повторюємо
        if ("429" in str(e) or "quota" in str(e).lower() or "rate
limit" in str(e).lower()) and retry_count < MAX_RETRIES:
            wait_time = RETRY_DELAY * (retry_count + 1)
            logger.info(f"□ Спробуємо ще раз через {wait_time}
сек... ({retry_count + 1}/{MAX_RETRIES})")
            time.sleep(wait_time)
            return make_gemini_request(chat_session,
user_message, retry_count + 1)
        raise

@csrf_exempt
@require_http_methods(["GET", "POST"])
def chat_index(request):
    if request.method == 'GET':
        return render(request, 'chat/index.html')

    if not gemini_model and not initialize_gemini():
        return JsonResponse({
            'error': 'AI сервіс недоступний. Спробуйте пізніше.',
            'status': 'service_unavailable'
        }, status=503)

    try:

        if request.content_type == 'application/json':
            data = json.loads(request.body)
        else:
            data = request.POST

        user_message = data.get('message', '').strip()
        if not user_message:
            return JsonResponse({
                'error': 'Порожнє повідомлення',
                'status': 'bad_request'
            }, status=400)


        chat_session = gemini_model.start_chat(history=[])


        response = make_gemini_request(chat_session,
user_message)

        return JsonResponse({
            'user_message': user_message,
            'bot_reply': response.text,
            'model': current_model_name,
            'requests_left': {
                'minute':
FREE_TIER_LIMITS[current_model_name]['requests_per_mi
nute'] - request_count_minute,
                'day':
FREE_TIER_LIMITS[current_model_name]['requests_per_da
y'] - request_count_day
            },
            'status': 'success'
        })

    except json.JSONDecodeError:
        logger.error("Невірний JSON у запиті")
        return JsonResponse({
            'error': 'Невірний формат запиту',
            'status': 'invalid_format'
        }, status=400)

    except Exception as e:
        logger.error(f"Помилка обробки запиту: {str(e)}")

        if "429" in str(e) or "quota" in str(e).lower() or "ліміт" in
str(e).lower():
            return JsonResponse({
                'error': 'Перевищено ліміти безкоштовного тарифу.
Будь ласка, спробуйте пізніше або оновіть свій план.',
                'status': 'rate_limit_exceeded',
                'model': current_model_name,
                'limits':
FREE_TIER_LIMITS.get(current_model_name, {}),
                'retry_after': RETRY_DELAY
            }, status=429)
        else:
            return JsonResponse({
                'error': 'Внутрішня помилка сервера',
                'status': 'server_error',
                'details': str(e)
            }, status=500)
from django.contrib import messages
from django.http import JsonResponse
```

```
from django.shortcuts import render, get_object_or_404,
redirect
from django.views.decorators.http import require_GET
from .models import ServicePackage, LeasingContract
from .forms import LeasingContractForm
from django.contrib.auth.decorators import login_required
from cars.models import Car
from django.utils.translation import gettext_lazy as _
from django.views.decorators.csrf import csrf_exempt  #
Змінив csrf_protect на csrf_exempt
from datetime import datetime, date, timedelta
from django.db.models import Q
from django.core.exceptions import ObjectDoesNotExist
from django.urls import reverse


def legalentities(request):
    return render(request, 'legalentities.html')


def individuals(request):
    return render(request, 'individuals.html')


@login_required
@csrf_exempt  # Виправив на csrf_exempt для тестування
def create_contract(request, car_code):
    car = get_object_or_404(Car, code=car_code)
    service_packages = ServicePackage.objects.all()
    selected_package = None
    error_message = None
    car_part = (car.price / 100) * 2
    calculated_prices = []

    for package in service_packages:
        total_price = package.price * 12 + car_part
        calculated_prices.append({
            'package': package,
            'total_price': round(total_price, 2),
            'monthly_payment': round(total_price / 12, 2),
        })

    if request.method == 'POST':
        form = LeasingContractForm(request.POST)
        if form.is_valid():
            contract = form.save(commit=False)
            contract.user = request.user
            contract.car = car

            # Додаткові перевірки дат
            today = date.today()
            if contract.start_date <= today:
                error_message = _("Дата початку повинна бути у
майбутньому.")
            elif contract.end_date <= contract.start_date:
                error_message = _("Дата завершення повинна бути
після дати початку.")
            elif (contract.end_date - contract.start_date).days < 30:
                error_message = _("Мінімальний термін лізингу -
30 днів.")
            else:
                # Перевірка на перетин дат
                overlapping = LeasingContract.objects.filter(
                    car=car
                ).filter(
                    Q(start_date__lte=contract.end_date) &
                    Q(end_date__gte=contract.start_date)
                ).exclude(user=request.user)

                if overlapping.exists():
                    error_message = _("Цей автомобіль вже
забронюваний на обраний період.")
                else:

                    months = ((contract.end_date.year -
contract.start_date.year) * 12 +
                              (contract.end_date.month -
contract.start_date.month))
                    if contract.end_date.day < contract.start_date.day:
                        months -= 1
                    months = max(months, 1)  # Мінімум 1 місяць

                    contract.total_price =
contract.service_package.price * months + car_part
                    contract.monthly_payment = contract.total_price /
months
                    contract.save()
                    messages.success(request, _("Контракт успішно
створено!"))
                    return redirect(reverse('users:profile'))
    else:
        form = LeasingContractForm()

    package_id = request.GET.get('package')
    if package_id:
        selected_package = get_object_or_404(ServicePackage,
id=package_id)
        form.fields['service_package'].initial = selected_package

    context = {
        'form': form,
        'car': car,
        'service_packages': service_packages,
        'selected_package': selected_package,
        'error_message': error_message,
        'calculated_prices': calculated_prices,
        'now': datetime.now(),
        'min_start_date': (date.today() +
timedelta(days=1)).strftime('%Y-%m-%d'),
        'min_end_date': (date.today() +
timedelta(days=31)).strftime('%Y-%m-%d')
    }
    return render(request, 'create_contract.html', context)


@login_required
def view_contracts(request):
    show_all = request.user.is_staff or request.user.is_superuser
    contracts = LeasingContract.objects.all() if show_all else
LeasingContract.objects.filter(user=request.user)
    contracts = contracts.select_related('car', 'user',
'service_package').order_by('-start_date')

    context = {
        'contracts': contracts,
        'page_title': _("Всі контракти лізингу"),
        'now': datetime.now(),
        'show_all': show_all
    }
    return render(request, 'contracts.html', context)


@login_required
def profile(request):
    show_all = request.user.is_staff or request.user.is_superuser
    contracts = LeasingContract.objects.all() if show_all else
LeasingContract.objects.filter(user=request.user)
    contracts = contracts.select_related('car', 'user',
'service_package').order_by('-start_date')
```

```
    context = {
        'contracts': contracts,
        'page_title': _("Особистий кабінет"),
        'now': datetime.now(),
        'show_all': show_all
    }
    return render(request, 'profile.html', context)


@require_GET
def get_package_details(request, package_id):
    try:
        package = ServicePackage.objects.get(id=package_id)
        data = {
            'id': package.id,
            'name': package.name,
            'price': float(package.price),
            'description': package.description,
            'services': [service.name for service in
package.services.all()],
        }
        return JsonResponse(data)
    except ObjectDoesNotExist:
        return JsonResponse({'error': _("Пакет не знайдено")},
status=404)


def leasing_services(request):
    service_packages =
ServicePackage.objects.prefetch_related('services').all()
    base_leasing = ServicePackage.objects.filter(name="Base
Leasing").first()
    base_services_count = base_leasing.services.count() if
base_leasing else 0

    for package in service_packages:
        package.base_services_count = base_services_count
        package.premium_services_count =
package.services.count() - base_services_count \
            if package.name != "Base Leasing" else 0

    context = {
        'service_packages': service_packages,
        'page_title': _("Послуги лізингу")
    }
    return render(request, 'leasing_services.html', context)
from django.utils import timezone
from decimal import Decimal
from django.contrib.auth.decorators import login_required
from django.contrib import auth, messages
from django.http import HttpResponseRedirect
from django.shortcuts import redirect, render
from django.urls import reverse

from leasing.models import LeasingContract
from users.forms import ProfileForm, UserLoginForm,
UserRegistrationForm


def login(request):
    if request.method == 'POST':
        form = UserLoginForm(data=request.POST)
        if form.is_valid():
            username = request.POST['username']
            password = request.POST['password']
            user = auth.authenticate(username=username,
password=password)
            if user:
                auth.login(request, user)
                messages.success(request, f"{username}, Ви
увійшли у акаунт")

                if request.POST.get('next', None):
                    return
HttpResponseRedirect(request.POST.get('next'))

                return HttpResponseRedirect(reverse('main:index'))
    else:
        form = UserLoginForm()

    context = {
        'title': 'Home - Авторизація',
        'form': form
    }
    return render(request, 'users/login.html', context)


def registration(request):
    if request.method == 'POST':
        form = UserRegistrationForm(data=request.POST)
        if form.is_valid():
            form.save()
            user = form.instance
            auth.login(request, user)
            messages.success(request, f"{user.username}, Ви
успішно створили аккаунт")
            return HttpResponseRedirect(reverse('main:index'))
    else:
        form = UserRegistrationForm()

    context = {
        'title': 'Home - Реєстрація',
        'form': form
    }
    return render(request, 'users/registration.html', context)


@login_required
def profile(request):
    if request.method == 'POST':
        form = ProfileForm(data=request.POST,
instance=request.user, files=request.FILES)
        if form.is_valid():
            form.save()
            messages.success(request, "Профіль успішно
оновлено")
            return HttpResponseRedirect(reverse('user:profile'))
    else:
        form = ProfileForm(instance=request.user)

    user = request.user
    contracts = LeasingContract.objects.filter(user=user)

    today_date = timezone.now().date()
    monthly_payments = []
    for contract in contracts:
        end_date = contract.end_date
        start_date = contract.start_date
        days_difference = (end_date - start_date).days
        monthly_payment = contract.total_price /
Decimal(days_difference / 30)
        if start_date > today_date:
            payment_info = {
                'contract': contract,
                'message': f"Перша оплата: {start_date} -
{round(monthly_payment, 2)} $"
            }
        else:
            payment_info = {
                'contract': contract,
```

```
            'message': f"Сума до сплати за цей місяць:
{round(monthly_payment, 2)} $"
        }
        monthly_payments.append(payment_info)
        contract.monthly_payment = round(monthly_payment, 2)

    context = {
        'user': user,
        'contracts': contracts,
        'monthly_payments': monthly_payments,
        'today_date': today_date,
        'form': form
    }

    return render(request, 'users/profile.html', context)


@login_required
def logout(request):
    messages.success(request, f"{request.user.username}, Ви
вийшли з аккаунту")
    auth.logout(request)
    return redirect(reverse('catalog:index'))
{% block content %}
<div id="aiChatModal" class="modal">
    <div class="modal-content">
        <span class="close"
onclick="closeAIChat()">&times;</span>
        <h4>Чат з AI-помічником</h4>
        <div id="chatHistory" style="height:300px; overflow-
y:auto; border:1px solid #ccc; padding:10px; margin-
bottom:10px;"></div>
        <div class="input-group">
            <input type="text" id="userInput" class="form-control"
placeholder="Введіть ваше запитання..."
onkeypress="handleKeyPress(event)">
            <button class="btn btn-primary"
onclick="sendMessage()">Надіслати</button>
        </div>
        <div class="chat-history-controls">
            <button class="btn btn-secondary"
onclick="toggleChatHistory()"
id="historyToggleBtn">Показати історію</button>
            <button class="btn btn-outline-danger"
onclick="clearChatHistory()">Очистити історію</button>
        </div>
    </div>
</div>

<div id="comparison-table" class="container my-4"
style="display:none;">
    <h3>Порівняння авто</h3>
    <div class="table-wrapper">
        <table class="table table-bordered">
            <thead>
                <tr>
                    <th>Авто</th>
                    <th>Ціна</th>
                    <th>Рік</th>
                    <th>Двигун</th>
                    <th>Паливо</th>
                </tr>
            </thead>
            <tbody id="comparison-body"></tbody>
        </table>
    </div>
    <button onclick="closeComparisonTable()" class="btn btn-
outline-danger mt-3">Закрити</button>
</div>

<div class="buttons-row">
    <div class="dropdown">
        <button class="btn btn-dark dropdown-toggle"
type="button" data-bs-toggle="dropdown" aria-
expanded="false">
            {% trans "Фільтри" %}
        </button>
        <form action="{% if request.GET.q %}{% url
"catalog:search" %}{% else %}{% url "catalog:index"%}{%
endif %}" method="get" class="dropdown-menu bg-dark"
data-bs-theme="dark">
            <div class="form-check text-white mx-3">
                <input class="form-check-input" type="checkbox"
name="discount" id="discountCheckbox" value="true" {% if
discount_filter %}checked{% endif %}>
                {% if request.GET.q %}
                    <input type="hidden" name="q"
value="{{ request.GET.q }}">
                {% endif %}
                <label class="form-check-label"
for="discountCheckbox">
                    {% trans "З і знижкою" %}
                </label>
            </div>
            <p class="text-white mx-3 mt-3">{% trans
"Сортування" %}</p>
            <div class="form-check text-white mx-3">
                <input class="form-check-input" type="radio"
name="order_by" id="flexRadioDefault1" value="default"
                {% if not request.GET.order_by or
request.GET.order_by == 'default' %}checked{% endif %}>
                <label class="form-check-label"
for="flexRadioDefault1">
                            {% trans "За замовчуванням" %}
                </label>
            </div>
            <div class="form-check text-white mx-3">
                <input class="form-check-input" type="radio"
name="order_by" id="flexRadioDefault2" value="price"
                {% if request.GET.order_by == 'price' %}checked{%
endif %}>
                <label class="form-check-label"
for="flexRadioDefault2">
                    {% trans "Ціна: від низької до високої" %}
                </label>
            </div>
            <div class="form-check text-white mx-3">
                <input class="form-check-input" type="radio"
name="order_by" id="flexRadioDefault3" value="-price"
                {% if request.GET.order_by == '-
price' %}checked{% endif %}>
                <label class="form-check-label"
for="flexRadioDefault3">
                    {% trans "Ціна: від високої до низької" %}
                </label>
            </div>
            <button type="submit" class="btn btn-primary mx-3
mt-3">{% trans "Застосувати" %}</button>
        </form>
    </div>

    <button onclick="showComparisonTable()" class="btn btn-
dark">Порівняти авто</button>
    <button onclick="showFavorites()" class="btn btn-
dark">Обране</button>
    <button onclick="closeFavorites()" class="btn btn-dark"
id="close-favorites">Закрити</button>
    <button onclick="openAIChat()" class="btn btn-dark">AI-
помічник</button>
</div>
```

```
<div class="container car-container">
  <div class="row row-cols-1 row-cols-md-3 g-4" id="car-list">
    {% for car in cars %}
    <div class="col" data-code="{{ car.code }}">
      <div class="card border-primary rounded custom-shadow">
        <a href="{% url 'cars:auto' car.code %}">
          <img src="{% static '/deps/images/cars/'|add:car.slug|add:'.jpeg' %}" class="card-img-top" alt="{{ car.name }}">
        </a>
        <div class="card-body">
          <div class="title-wrapper">
            <a href="{% url 'cars:auto' car.code %}" class="text-decoration-none">
              <p class="card-title fw-bold">{{ car.name }}</p>
            </a>
            <div class="favorite-actions">
              <i class="favorite-icon bi {% if car.code in favorite_cars %}bi-heart-fill{% else %}bi-heart{% endif %}"
                 onclick="toggleFavorite('{{ car.code }}')"
                 id="fav-{{ car.code }}"></i>
              <button class="remove-btn" onclick="removeFavorite('{{ car.code }}')">×</button>
            </div>
          </div>
          <p class="card-text text-truncate">
            {% trans "Рік" %}: {{ car.year }}.
            {% trans "Двигун" %}: {{ car.engine_capacity }} л,
            {{ car.get_fuel_type_display }}.
          </p>

<p><strong>{{ car.discounted_price|default:car.price }} $</strong></p>
          <button onclick="addToCompare('{{ car.code }}', '{{ car.name }}', '{{ car.price }}', '{{ car.year }}', '{{ car.engine_capacity }}', '{{ car.get_fuel_type_display }}', '{{ car.discounted_price|default:" }}')" class="btn btn-outline-primary mt-2">{% trans "Додати до порівняння" %}</button>
        </div>
      </div>
    </div>
    {% endfor %}
  </div>
</div>

<script>
  let selectedCars = [];
  let favoriteCars = JSON.parse(localStorage.getItem('favorites')) || [];
  let isFavoritesView = false;
  let chatHistory = JSON.parse(localStorage.getItem('chatHistory')) || [];
  let isViewingFullHistory = false;
  let currentSessionMessages = [];
  let isFirstChatOpen = true;

  function openAIChat() {
    document.getElementById('aiChatModal').style.display = 'block';


    if (isFirstChatOpen && currentSessionMessages.length === 0) {
      const welcomeMessage = {
        type: 'ai',
        content: "□ Вітаємо! Ви потрапили на нашу платформу з інформацією про доступні автомобілі, доступні типи лізингу та актуальні законодавство України щодо лізингу. □\n\nУ нас є великий вибір автомобілів, доступних для лізингу □\n\nБажаєте дізнатися більше про:\n- кожен тип лізингу\n- необхідні документи\n- законодавчі зміни\n- правила лізингу в Україні?\n\nПросто напишіть ваше запитання, і ми надамо всю необхідну інформацію!",
        buttons: true
      };

      currentSessionMessages.push(welcomeMessage);
      chatHistory.push(welcomeMessage);
      localStorage.setItem('chatHistory', JSON.stringify(chatHistory));
      localStorage.setItem('currentSessionMessages', JSON.stringify(currentSessionMessages));

      isFirstChatOpen = false;
    }

    displayCurrentSession();
    const chatHistoryDiv = document.getElementById('chatHistory');
    chatHistoryDiv.scrollTop = chatHistoryDiv.scrollHeight;
  }

  function closeAIChat() {
    document.getElementById('aiChatModal').style.display = 'none';
  }

  function handleKeyPress(event) {
    if (event.key === 'Enter') {
      sendMessage();
    }
  }

  function sendMessage() {
    const userInput = document.getElementById('userInput');
    const message = userInput.value.trim();
    if (!message) return;

    const chatHistoryDiv = document.getElementById('chatHistory');
    const userMessage = { type: 'user', content: message };

    currentSessionMessages.push(userMessage);
    chatHistory.push(userMessage);

    localStorage.setItem('chatHistory', JSON.stringify(chatHistory));
    localStorage.setItem('currentSessionMessages', JSON.stringify(currentSessionMessages));

    chatHistoryDiv.innerHTML += `
      <div class="chat-message user-message">
        <strong>Ви:</strong> ${message}
      </div>
    `;

    userInput.value = '';
    chatHistoryDiv.scrollTop = chatHistoryDiv.scrollHeight;

    const typingIndicator = document.createElement('div');
    typingIndicator.id = 'typing-indicator';
    typingIndicator.className = 'typing-indicator';
    typingIndicator.innerHTML = '<em>AI набирає
```

```
відповідь...</em>';
        chatHistoryDiv.appendChild(typingIndicator);
        chatHistoryDiv.scrollTop = chatHistoryDiv.scrollHeight;

        function getCookie(name) {
            let cookieValue = null;
            if (document.cookie && document.cookie !== '') {
                const cookies = document.cookie.split(';');
                for (let i = 0; i < cookies.length; i++) {
                    const cookie = cookies[i].trim();
                    if (cookie.substring(0, name.length + 1) ===
(name + '=')) {
                        cookieValue =
decodeURIComponent(cookie.substring(name.length + 1));
                        break;
                    }
                }
            }
            return cookieValue;
        }
        const csrftoken = getCookie('csrftoken');

        fetch('/gemini-chat/', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRFToken': csrftoken
            },
            body: JSON.stringify({ message: message })
        })
        .then(response => {
            if (!response.ok) {
                throw new Error('Помилка мережі');
            }
            return response.json();
        })
        .then(data => {
            const indicator = document.getElementById('typing-
indicator');
            if (indicator) indicator.remove();

            if (data.error) {
                throw new Error(data.error);
            }

            let response = data.bot_reply || data.reply || "Не вдалося
отримати відповідь";
            response = response.replace(/\*/g, '');

            const aiMessage = { type: 'ai', content: response };

            currentSessionMessages.push(aiMessage);
            chatHistory.push(aiMessage);

            localStorage.setItem('chatHistory',
JSON.stringify(chatHistory));
            localStorage.setItem('currentSessionMessages',
JSON.stringify(currentSessionMessages));

            chatHistoryDiv.innerHTML += `
                <div class="chat-message ai-message">
                    <strong>AI:</strong> ${response}
                </div>
            `;
            chatHistoryDiv.scrollTop =
chatHistoryDiv.scrollHeight;
        })
        .catch(error => {
            console.error('Помилка:', error);
            const indicator = document.getElementById('typing-
indicator');
            if (indicator) indicator.remove();

            const fallbackResponse = "Вибачте, сталася помилка.
Будь ласка, спробуйте пізніше.";
            const aiMessage = { type: 'ai', content:
fallbackResponse };

            currentSessionMessages.push(aiMessage);
            chatHistory.push(aiMessage);

            localStorage.setItem('chatHistory',
JSON.stringify(chatHistory));
            localStorage.setItem('currentSessionMessages',
JSON.stringify(currentSessionMessages));

            chatHistoryDiv.innerHTML += `
                <div class="chat-message ai-message">
                    <strong>AI:</strong> ${fallbackResponse}
                </div>
            `;
            chatHistoryDiv.scrollTop =
chatHistoryDiv.scrollHeight;
        });
    }

    function sendPredefinedMessage(message) {
        document.getElementById('userInput').value = message;
        sendMessage();
    }

    function displayCurrentSession() {
        const chatHistoryDiv =
document.getElementById('chatHistory');
        chatHistoryDiv.innerHTML = '';

        currentSessionMessages.forEach(msg => {
            let messageContent = msg.content;
            if (msg.buttons) {
                messageContent += `
                    <div class="car-buttons-row">
                        <button class="car-btn"
onclick="showAvailableCars()">Авто у наявності</button>
                        <button class="car-btn"
onclick="showLeasingTypes()">Типи лізингу</button>
                        <button class="car-btn"
onclick="sendPredefinedMessage('Законодавство України
про лізинг')">Законодавство України про лізинг</button>
                    </div>
                `;
            }

            const messageClass = msg.type === 'user' ? 'user-
message' :
                                 msg.content.includes("□ Вітаємо!") ?
'welcome-message' : 'ai-message';
            const sender = msg.type === 'user' ? 'Ви:' : 'AI:';

            chatHistoryDiv.innerHTML += `
                <div class="chat-message ${messageClass}">
                    <strong>${sender}</strong> ${messageContent}
                </div>
            `;
        });
    }

    function displayFullHistory() {
        const chatHistoryDiv =
document.getElementById('chatHistory');
```

```
        chatHistoryDiv.innerHTML = '';

        chatHistory.forEach(msg => {
            let messageContent = msg.content;
            if (msg.buttons) {
                messageContent += `
                    <div class="car-buttons-row">
                        <button class="car-btn"
onclick="showAvailableCars()">Авто у наявності</button>
                        <button class="car-btn"
onclick="showLeasingTypes()">Типи лізингу</button>
                        <button class="car-btn"
onclick="sendPredefinedMessage('Законодавство України
про лізинг')">Законодавство України про лізинг</button>
                    </div>
                `;
            }
            const messageClass = msg.type === 'user' ? 'user-
message' :
                                 msg.content.includes("□ Вітаємо!") ?
'welcome-message' : 'ai-message';
            const sender = msg.type === 'user' ? 'Ви:' : 'AI:';

            chatHistoryDiv.innerHTML += `
                <div class="chat-message ${messageClass}">
                    <strong>${sender}</strong> ${messageContent}
                </div>
            `;
        });
    }

    function toggleChatHistory() {
        const chatHistoryDiv =
document.getElementById('chatHistory');
        const toggleBtn =
document.getElementById('historyToggleBtn');

        if (isViewingFullHistory) {
            displayCurrentSession();
            toggleBtn.textContent = 'Показати історію';
        } else {
            displayFullHistory();
            toggleBtn.textContent = 'Поточний чат';
        }

        isViewingFullHistory = !isViewingFullHistory;
        chatHistoryDiv.scrollTop = chatHistoryDiv.scrollHeight;
    }

    function clearChatHistory() {
        if (confirm('Ви впевнені, що хочете очистити історію
чатів?')) {
            chatHistory = [];
            currentSessionMessages = [];
            localStorage.setItem('chatHistory',
JSON.stringify(chatHistory));
            localStorage.setItem('currentSessionMessages',
JSON.stringify(currentSessionMessages));
            isFirstChatOpen = true;
            displayCurrentSession();
            if (isViewingFullHistory) {
                toggleChatHistory();
            }
        }
    }
```