

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

## Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ НАДАННЯ ВЛАСНИХ  
АВТОМОБІЛІВ У ПРОКАТ  
SOFTWARE FOR CAR RENTAL FROM OWNERS

Виконав: студент(ка) 4 курсу, групи КНТ-128  
Спеціальності 121 Інженерія програмного  
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)  
Інженерія програмного забезпечення

Шевченко І.Д.

(прізвище та ініціали)

Керівник Казурова А.Є.

(прізвище та ініціали)

Рецензент Голуб Т.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»  
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ  
Кафедра програмних засобів  
Ступінь вищої освіти бакалавр  
Спеціальність 121 Інженерія програмного забезпечення  
(код і найменування)  
Освітня програма (спеціалізація) Інженерія програмного забезпечення  
(назва освітньої програми (спеціалізації))

**ЗАТВЕРДЖУЮ**

Завідувач кафедри ПЗ, д.т.н, проф.  
С.О. Субботін  
“      ”                                  2022 року

**З А В Д А Н Н Я**

**НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)**

Шевченка Іллі Денисовича  
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення надання власних автомобілів у прокат  
Software for Car Rental from Owners

керівник проєкту (роботи) Казурова Аліна Євгенівна, к.т.н., доцент,  
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) червень 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

## 6. Консультанти розділів проєкту (роботи)

| Розділ              | Прізвище, ініціали та посада консультанта | Підпис, дата   |                           |
|---------------------|-------------------------------------------|----------------|---------------------------|
|                     |                                           | завдання видав | прийняв виконане завдання |
| 1-4 Основна частина | Казурова А.Є., доцент                     |                |                           |
| Нормоконтролер      | Дейнега Л.Ю., ст. викладач                |                |                           |
|                     |                                           |                |                           |
|                     |                                           |                |                           |

7. Дата видачі завдання «4» квітня 2022 року.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проєкту (роботи)                                             | Строк виконання етапів проєкту (роботи) | Примітка     |
|-------|--------------------------------------------------------------------------------------|-----------------------------------------|--------------|
| 1     | Постановка завдання роботи                                                           | 1 тиждень                               | Завдання, ТЗ |
| 2     | Аналіз предметної області                                                            | 2 тиждень                               | Розділ 1     |
| 3     | Розробка архітектури програми                                                        | 3 тиждень                               | Розділ 2     |
| 4     | Розробка програми                                                                    | 4-5 тижні                               | Розділ 3     |
| 5     | Тестування та експериментальне дослідження програми                                  | 6 тиждень                               | Розділ 4     |
| 6     | Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування | 7 тиждень                               | Додатки      |
| 7     | Захист роботи                                                                        | 8 тиждень                               |              |
|       |                                                                                      |                                         |              |
|       |                                                                                      |                                         |              |
|       |                                                                                      |                                         |              |
|       |                                                                                      |                                         |              |
|       |                                                                                      |                                         |              |
|       |                                                                                      |                                         |              |

Студент(ка)

\_\_\_\_\_  
( підпис )

Шевченко І.Д.  
(прізвище та ініціали)

Керівник проєкту (роботи)

\_\_\_\_\_  
( підпис )

Казурова А.Є.  
(прізвище та ініціали)

## РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 80 с., 22 рис., 2 табл., 3 дод., 11 джерел.

АВТОМОБІЛЬ, ПРОКАТ АВТОМОБІЛЯ, ВЛАСНИК АВТОМОБІЛЯ, ВЕБЗАСТОСУНОК, ВЕБФРЕЙМВОРК.

Об'єктом роботи є процес прокату автомобілів. Предметом роботи є програмне забезпечення надання власних автомобілів у прокат.

Мета роботи – розробити програмне забезпечення надання власних автомобілів у прокат для підтримки взаємодії між власниками автомобілів і орендарями.

Матеріали, методи та технічні засоби: мова програмування Python, система керування базами даних MySQL, фреймворк для розробки вебзастосунків Django.

Результати. Основні отримані результати дипломної роботи охоплюють етапи розробки програми, включаючи її проєктування на основі аналізу предметної області, створену базу даних, визначені вимоги, реалізовані структурні одиниці програми і саме реалізоване програмне забезпечення.

Висновки. Розроблено вебзастосунок для надання можливості власникам автомобілів виставити на прокат власний автомобіль та впорядковувати такі прокати, а клієнтам відповідно брати автомобілі на прокат у фізичних осіб, а не у компаній, керуючи ними самостійно або з водіями. Програмне забезпечення таким чином дозволяє впорядкувати сферу індивідуальних прокатів і створити можливість тимчасового заробітку власника автомобілів у час, коли автомобіль простоє і не використовується.

Галузь використання. Прокат автомобілів, де автомобілі надаються їх власниками.

## **ABSTRACT**

Explanatory note to the diploma qualifying work of the bachelor: 80 pages, 22 figures, 2 tables, 3 appendixes, 11 sources.

CAR, CAR RENTAL, CAR OWNER, WEB APPLICATION, WEB FRAMEWORK.

The object of the work is the process of car rental. The subject of the work is the software for providing own cars for rent.

The purpose of the work is to develop software for renting cars from owners to support interaction between car owners and tenants.

Materials, methods and technical means: Python programming language, MySQL database management system, Django web application development framework.

Results. The main results of the work cover the stages of program development, including its design based on the analysis of the subject area, the created database, the defined requirements, the implemented structural units of the software and the software itself.

Conclusions. Web application has been developed to enable car owners to rent their own car and arrange such rentals and to enable customers to rent cars from individuals, not from companies, driving them alone or with drivers. The software thus allows users to streamline the scope of individual rentals and create the possibility of temporary earnings of the car owner at a time when the car is idle and unused.

Field of use. Car rentals, where cars are provided by their owners.

## ЗМІСТ

|                                                                 |    |
|-----------------------------------------------------------------|----|
|                                                                 | С. |
| Перелік скорочень та умовних познач .....                       | 7  |
| Вступ .....                                                     | 8  |
| 1 Аналіз предметної області .....                               | 10 |
| 1.1 Проблема прокату автомобілів, що надаються власниками ..... | 10 |
| 1.2 Аналіз програмних аналогів .....                            | 11 |
| 1.3 Висновки за розділом 1 .....                                | 14 |
| 2 Розробка архітектури програми .....                           | 16 |
| 2.1 Вибір засобів розробки програми .....                       | 16 |
| 2.2 Функціональні вимоги до програмного забезпечення .....      | 17 |
| 2.3 Проектування бази даних .....                               | 19 |
| 2.4 Висновки за розділом 2 .....                                | 31 |
| 3 Розробка програми .....                                       | 32 |
| 3.1 Основні принципи організації розробки програми .....        | 32 |
| 3.2 Опис реалізації програми .....                              | 33 |
| 3.3 Висновки за розділом 3 .....                                | 38 |
| 4 Експлуатація та тестування програми .....                     | 39 |
| 4.1 Призначення програми .....                                  | 39 |
| 4.2 Умови виконання програми .....                              | 39 |
| 4.3 Виконання програми .....                                    | 39 |
| 4.4 Тестування програми .....                                   | 44 |
| 4.5 Висновки за розділом 4 .....                                | 44 |
| Висновки .....                                                  | 45 |
| Перелік джерел посилання .....                                  | 46 |
| Додаток А Технічне завдання .....                               | 48 |
| Додаток Б Текст програми .....                                  | 52 |
| Додаток В Слайди презентації .....                              | 74 |

## **ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК**

VIN – Vehicle Identification Number;

БД – база даних;

ПЗ – програмне забезпечення.

## ВСТУП

Ринок спільного використання автомобілів, коли клієнт може взяти автомобіль на прокат в місті, а потім залишити його, щоб інший клієнт скористався ним, при цьому реалізуючи цей процес без необхідності довгої фіксації, активно розвивається: за оцінками експертів, до 2024 року цей ринок перевищить 11 мільярдів доларів, але, незважаючи на такий величезний потенціал зростання, залишаються значні проблеми [1]-[2]. Ці проблеми зокрема пов'язані з забезпеченням чистоти в автомобілі, адже поїздки можуть бути короткими, а прибирання між клієнтами не виконується. Але найголовніша проблема пов'язана з тим, що не у всіх містах світу такі сервіси присутні, а це призводить до того, що потрібно користатися традиційними сервісами прокату. Це визначає обмеження пунктів прокату, тобто взяти автомобіль можна тільки в чітко визначених локаціях. Автомобілі, що надаються в прокат, зазвичай є типовими, знайти особливу модель важко. Все це вказує на те, що такі сервіси не для всіх клієнтів є вирішенням їх проблем.

Послуги прокату автомобілів є достатньо затребуваними, але надаються вони централізованими компаніями, у той же час є простір для розширення прокату за рахунок індивідуальних власників. Фактично вони вже зараз можуть надавати свої автомобілі за оголошеннями, знайомим, формуючи таким чином простір для виконання прокатів. Але потрібно мати інструмент для взаємодії між власниками автомобілів і потенційними клієнтами, інакше все це залишиться на рівні окремих випадків, а не індустрії. За результатами попереднього аналізу було виявлено, що подібні програмні інструменти є в деяких країнах, але в Україні відсутні. Тому важливо створити програмне забезпечення (ПЗ) для підтримки саме індивідуальних власників автомобілів, яке б забезпечило вирішення окресленої проблеми.

Метою дипломної роботи була визначена розробка ПЗ надання власних автомобілів у прокат для підтримки взаємодії між власниками автомобілів і орендарями.

## **1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

### **1.1 Проблема прокату автомобілів, що надаються власниками**

Оскільки кількість транспортних засобів у всьому світі зростає в геометричній прогресії, багато міст відчувають великі затори, це створює проблему для користування власним автомобілем: з обмеженою дорожньою інфраструктурою та громадським транспортом країни, що розвиваються, особливо вразливі до екстремальних заторів, згідно з щорічним індексом трафіку TomTom, 13 із 20 найгірших міст для поїздок на роботу розташовані в країнах з ринковою економікою, включаючи Китай, Бразилію, Туреччину та Мексику [2]. Проте на даний момент в цих переліках став активно з'являтися Київ. Збільшення кількості автомобілів призводить до того, що час, який жителі міста проводять в корках, весь час збільшується. І це збільшення можна назвати драматичним.

Велика кількість жителів міста користується автомобілем в певні моменти часу, а в інші моменти ці автомобілі простоюють. Автомобілі перебувають на стоянках 95 % часу свого життя: ця цифра широко використовується, щоб показати, наскільки неефективною є парадигма власності на автомобіль [3]-[4].

До того ж проблема змінилась в Україні під час війни, з погіршенням доступності пального. Навпаки вивільнилась велика кількість автомобілів, які стали використовуватися рідко. Все це підкреслює ту проблему, що ефективність використання транспорту в різних ситуаціях диктує те, що власний автомобіль може бути корисно віддати в прокат іншій людині.

У стандартних сервісах прокату автомобілів стороння компанія володіє парком транспортних засобів, які клієнти можуть використовувати в обмін на плату, при цьому компанія утримує автопарк і часто оплачує паливе, на відміну від такого однорангового прокату автомобілів, у дворанговій системі приватні особи здають свої транспортні засоби в прокат водіям, а компанія просто виступає як посередник, вона не платить за паливо,

не надає транспортні засоби та не проводить будь-яке технічне обслуговування, це дозволяє розширювати охоплення набагато швидше, ніж у однорангових сервісів прокату автомобілів, оскільки все, що в такому випадку потрібно, щоб вийти на ринок, – це достатня кількість зацікавлених власників і орендарів транспортних засобів [5].

Подібні сервіси програми працюють на одній основі: власник вносить автомобіль на платформу, надаючи фотографії, ціни та іншу ключову інформацію про автомобіль, зокрема його базові характеристики, його серійний номер, що має назву Vehicle Identification Number (VIN), користувач платформи переглядає застосунок, щоб знайти автомобіль, який він бажає орендувати, вказує, на який термін він бажає це зробити, а потім просить забронювати автомобіль, тоді власник автомобіля має погодитися з цим замовленням, підтверджуючи його, далі клієнт їде до власника, бере автомобіль і їде в дорогу, після завершення повертає автомобіль [5]-[6].

Саме застосунки, які побудовані на цьому принципі, і було розглянуто в даному розділі далі детальніше для того, щоб виявити функціональні особливості подібних програм і визначити можливості, якими повинна володіти програма, яка розробляється, в наступному розділі.

## **1.2 Аналіз програмних аналогів**

Першим з аналогів програми, що розробляється в роботі, було розглянуто вебзастосунок Turo (рис. 1.1) [7]. Це рішення загалом реалізовано як вебзастосунок, що дозволяє широкій кількості користувачів працювати з програмою і без авторизації і реєстрації проте також воно доступно і як застосунок для мобільних операційних систем.

Turo надає можливість виставити власний автомобіль для прокату, встановлюючи для нього набір характеристик, визначаючи вартість такого прокату. Надається можливість залишити відгук про автомобіль після прокату, а також про власника автомобіля. Саме відгуки дозволяють

орієнтуватися користувачам в програмі, обираючи потрібний автомобіль і оцінюючи, на скільки правдиву інформацію надано в оголошенні.

Пошук автомобілів включає розширену кількість фільтрів, які охоплюють як місто, де клієнт бажає взяти автомобіль у прокат, дати прокату, так і характеристики автомобіля, зокрема рік виробництва, кількість місць, категорію, тип коробки передач.

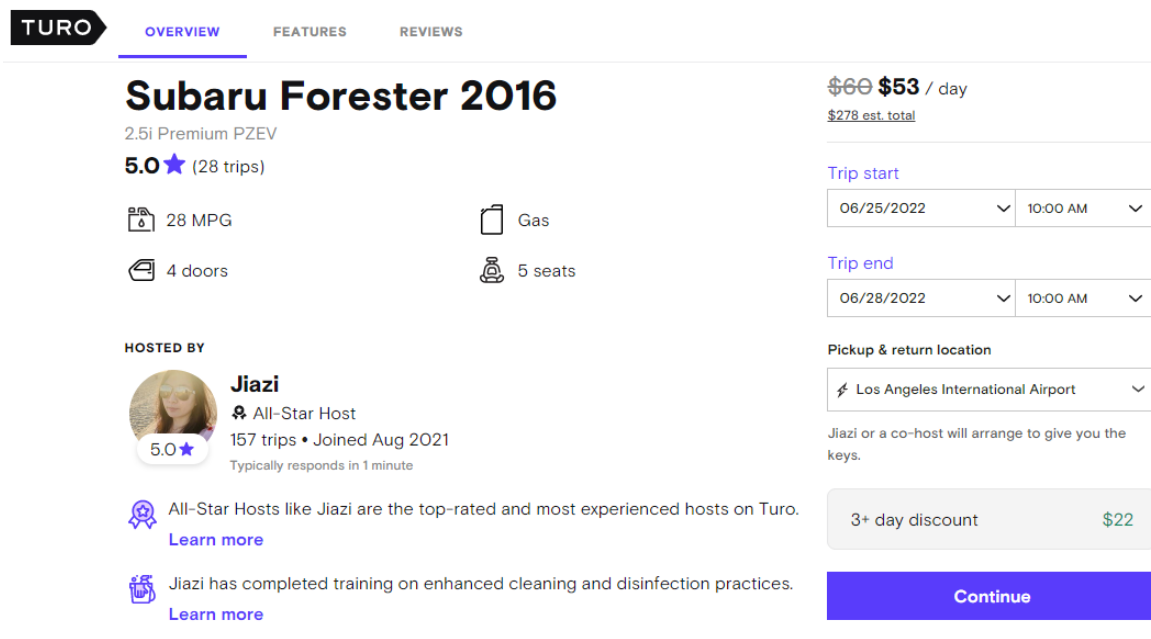


Рисунок 1.1 – Інтерфейс вебзастосунку Turo

Ще одним подібним застосунком є Rentben (рис. 1.2) [8]. Він влаштований за подібною логікою, проте реалізує прокат автомобілів від їх власників в Угорщині. Використання обмежене однією країною.

Пошук автомобілів у програмі можна виконувати як за базовою формою, що включає місто і строки оренди, так і за розширеним варіантом, який дозволяє встановлювати фільтри для результатів пошуку. Rentben також дозволяє створювати відгуки і виставляти оцінки.

Загалом слід констатувати, що ці застосунки є дуже подібними між собою, вони мають різну організацію інтерфейсу, різні конкретні фільтри пошуку, дещо різні характеристики автомобілів в оголошеннях, але за

загальними функціями вони достатньо подібні, що дозволяє проаналізовані таким чином функціональні можливості виділити для реалізації власної програми.

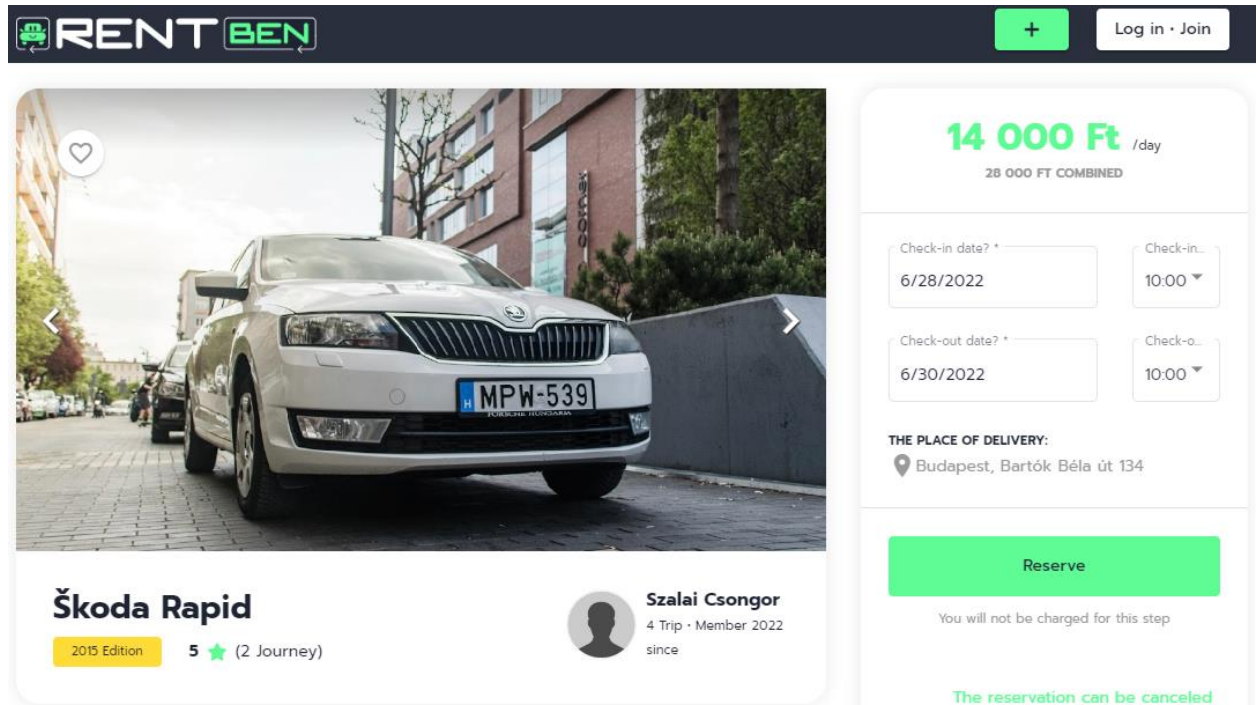


Рисунок 1.2 – Інтерфейс вебзастосунку Rentben

За результатами проведеного огляду програмних аналогів було виконано порівняння розглянутих аналогів і виділено особливості виконуваної розробки (табл. 1.1).

Власна розробка в таблиці позначена як OwnRentaler. Її потенційною країною використання є Україна, адже вона направлена на те, щоб сервіси дворангового прокату автомобілів запрацювали саме в Україні, де вони відсутні, а існують тільки дошки оголошень, на яких окремі власники можуть виставляти таким чином свої автомобілі в прокат. Відповідно немає адекватних засобів пошуку, формування замовлень, що не дозволяє розглядати такі вебсайти як повноцінний інструмент підтримки такого прокату автомобілів.

Особливостями програми, що розробляється, є також можливість визначення водіїв, які будуть керувати автомобілем, якщо клієнт бажає

отримати таку послугу, та можливість внесення дефектів автомобілів, щоб клієнт чіткіше розумів стан автомобіля до початку прокату. Ці можливості відсутні у аналогів.

Таблиця 1.1 – Порівняльна характеристика програмних аналогів з виконуваною розробкою

| Критерій                                    | Turo         | Rentben  | OwnRentaler                  |
|---------------------------------------------|--------------|----------|------------------------------|
| Країна дії                                  | Фактично США | Угорщина | Україна, може бути розширено |
| Розширена кількість фільтрів пошуку         | Так          | Так      | Так                          |
| Реєстрація власників автомобілів            | Так          | Так      | Так                          |
| Відгуки                                     | Так          | Так      | Так                          |
| Визначення водіїв для керування автомобілем | Ні           | Ні       | Так                          |
| Визначення дефектів автомобілів             | Ні           | Ні       | Так                          |

### 1.3 Висновки за розділом 1

За результатами огляду проблеми прокату автомобілів загалом було визначено підпроблему прокату автомобілів, що надаються їх власниками. Цей варіант прокату розглянуто як дворанговий спосіб і порівняно з одноранговим. Проаналізовано програми, які для такого дворангового прокату можуть застосовуватися: Turo, Rentben. Виділено особливості програми, яка розробляється, порівняно її з цими аналогами.

Для досягнення мети в роботі визначено завдання:

– аналіз предметної області;

- проєктування бази даних (БД);
- проєктування ПЗ надання власних автомобілів у прокат;
- програмна реалізація ПЗ надання власних автомобілів у прокат.

## 2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

### 2.1 Вибір засобів розробки програми

Вибір засобів розробки першим кроком включав вибір мови програмування для реалізації програми. Програму необхідно було реалізувати у вигляді вебзастосунку. Популярними мовами для розв’язання цієї задачі є Python [9]-[11] і Java (табл. 2.1).

Таблиця 2.1 – Вибір мови програмування для реалізації програми

| Критерій                                  | Python                                                              | Java        |
|-------------------------------------------|---------------------------------------------------------------------|-------------|
| Час виконання етапу програмної реалізації | Коротший за рахунок динамічної типізації та простоти створення коду | Довший      |
| Наявність фреймворку веброзробки          | Так, Django                                                         | Так, Spring |
| Синтаксис мови програмування              | Простіший                                                           | Складніший  |
| Рівень складності вивчення                | Менший                                                              | Більший     |

Обрано мову програмування Python, враховуючи, що вона дозволяє швидше виконувати етап програмної реалізації, відповідно це дозволяє дотримуватися часових обмежень розробки за календарним планом, синтаксис мови простіший, вона має менший рівень складності при вивченні, відповідно, якщо будуть використовуватися не відомі спочатку можливості, то вивчити їх буде простіше, а значить знову ж таки швидше. Також для пришвидшення розробки було вирішено використовувати фреймворк веброзробки Django.

## 2.2 Функціональні вимоги до програмного забезпечення

Загальний набір функціональних вимог до програми включає наступні:

- створення оголошення про прокат власного автомобіля;
- замовлення прокату автомобіля;
- редагування замовлення прокату автомобіля;
- підтвердження замовлення;
- скасування замовлення;
- керування станом замовлення;
- створення відгуку про автомобіль клієнтом;
- створення відгуку про клієнта власником автомобіля;
- перегляд відгуків про клієнта;
- перегляд відгуків про автомобіль;
- пошук автомобіля для прокату;
- перегляд замовлень прокату автомобіля власником;
- додавання водіїв для керування автомобілем;
- вилучення водіїв від керування автомобілем;
- перегляд автомобілів, якими може керувати водій;
- перегляд замовлень прокатів автомобілів, якими може керувати водій;
- авторизація, реєстрація користувачів;
- додавання дефектів автомобіля;
- редагування дефектів автомобіля;
- визначення вартості прокату автомобіля на період часу;
- змінювання вартості прокату автомобіля;
- визначення періоду доступності автомобіля для прокату;
- верифікація документів клієнта;
- перегляд оголошень власника;
- архівування оголошення;

– надсилання повідомлень.

Вимоги були використані для моделювання сценаріїв використання програми її користувачами, до яких віднесено клієнта (рис. 2.1), водія та власника автомобіля.

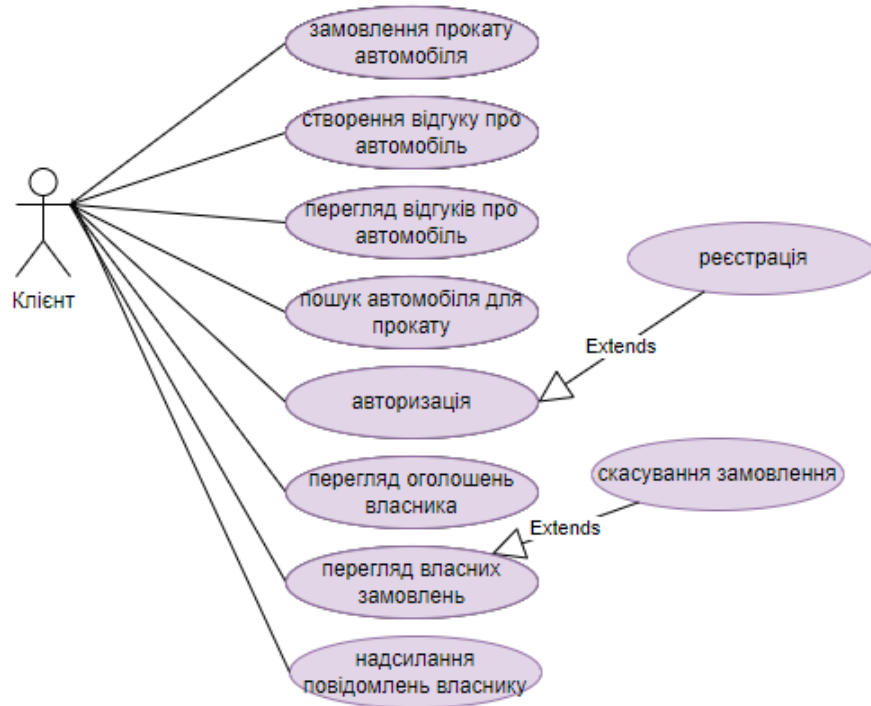


Рисунок 2.1 – Діаграма прецедентів клієнта

Водій є користувачем, якому дозволено керувати певні автомобілі їх власниками на випадок, якщо клієнт не бажає або не вміє керувати автомобілем самостійно (рис. 2.2).

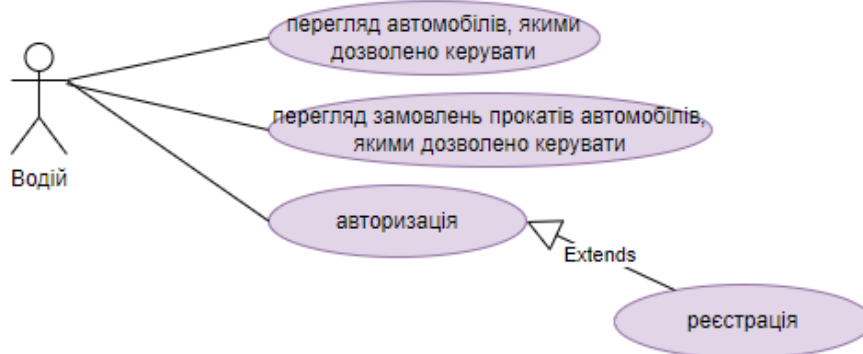


Рисунок 2.2 – Діаграма прецедентів водія

Сценарії використання програми власником автомобіля представлені на рис. 2.3.

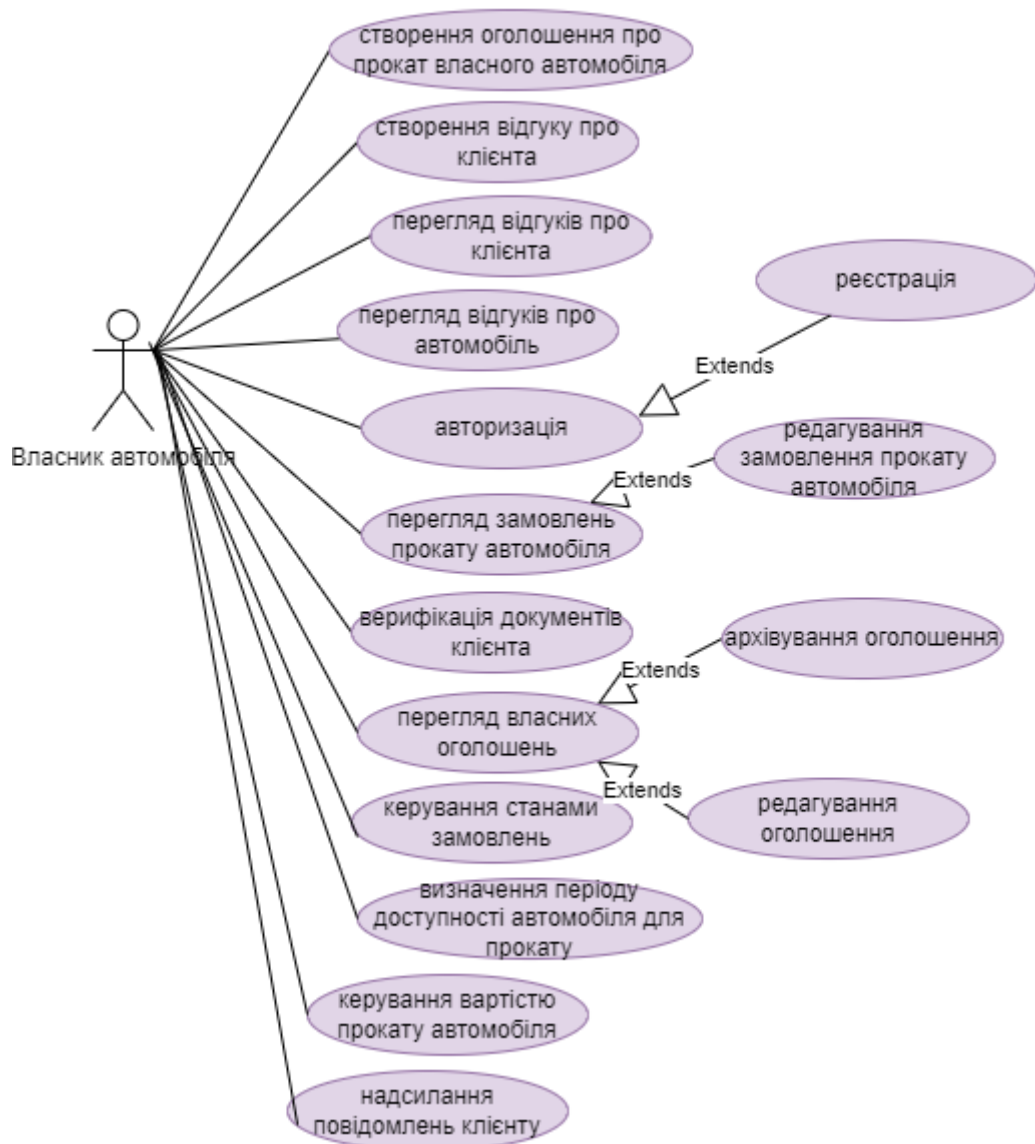


Рисунок 2.3 – Діаграма прецедентів водія автомобіля

## 2.3 Проєктування бази даних

Для створення БД використано систему керування БД MySQL. Це рішення обґрунтовано тим, що це система керування, яка активно використовується на практиці, є достатньо надійною, а до того ж є безкоштовною у використанні, що важливо для виконання розробки в межах дипломної роботи.

Оскільки робота користувачів у програмі відбувається з автомобілями, і кожен власник автомобіля має внести свій, то в програмі потрібен каталог автомобілів. Цей каталог повинен включати виробників і моделі автомобілів, які виробляються кожним виробником. Дані каталогу в результаті мають використовуватися для внесення автомобілів власника, а також для пошуку, адже це фактично один зі зручних способів категоризації автомобілів.

Сутність виробника автомобіля визначається в БД таблицею `Producer` з полями:

- `id` – ідентифікатор компанії виробника автомобілів;
- `prod_name` – назва компанії виробника;
- `characteristics` – опис компанії виробника, зокрема певних особливостей за необхідності;
- `country` – країна, в якій знаходиться виробник;
- `website` – вебсайт компанії виробника автомобілів.

Сутність моделі автомобіля визначається в БД таблицею `Machinemodel` з полями:

- `id` – ідентифікатор моделі автомобіля;
- `producer_id` – ідентифікатор виробника автомобілів цієї моделі;
- `model_name` – назва моделі автомобіля;
- `characteristics` – опис особливостей моделі автомобіля;
- `places` – кількість місць у автомобілях цієї моделі;
- `doors` – кількість дверей у автомобілях цієї моделі;
- `body_style` – тип кузова автомобіля;
- `length` – довжина автомобіля цієї моделі;
- `height` – висота автомобіля цієї моделі;
- `width` – ширина автомобіля цієї моделі;
- `website` – вебсайт з інформацією про модель;

Поле таблиці `producer_id` є зовнішнім ключем з таблиці `Producer`. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох»

між таблицями `Producer` і `Machine`: один виробник може виготовляти автомобілі різних моделей, проте кожна модель автомобіля виготовляється конкретною компанією виробником.

Сам автомобіль, який власник хоче віддати в прокат (конкретний автомобіль власника), визначається в БД таблицею `Machine`. Дані в цій таблиці фактично представляють оголошення, яким власник виставляє на прокат свій автомобіль. Поля таблиці:

- `id` – ідентифікатор автомобіля;
- `machinemodel_id` – модель автомобіля;
- `transmission` – тип коробки передач, яка використовується в автомобілі: винесений в автомобіль, а не в модель автомобіля через те, що одна модель автомобіля може мати декілька варіантів з різними коробками передач;
- `drive` – тип приводу автомобіля;
- `engine` – двигун автомобіля;
- `fuel_type` – тип пального, який використовується автомобілем (бензин, дизель, електрика);
- `fuel_consumption` – споживання пального на 100 кілометрів;
- `tires` – шини, які встановлені на автомобілі в поточний момент;
- `vin` – VIN-номер автомобіля: це унікальний номер, але він не був використаний в якості ключа, оскільки цей номер вноситься власником автомобіля, а не яким-небудь менеджером, відповідно, якщо власник введе номер некоректно, то це призведе до того, що інший користувач не зможе внести свій автомобіль під коректним номером, тому що такий запис вже зроблено;
- `conditioner` – тип системи кондиціонування автомобіля;
- `owner_id` – власник автомобіля;
- `city` – місто, де знаходиться автомобіль;
- `address` – адреса, за якою зазвичай знаходиться автомобіль або за якою він віддається в прокат;

- flexible – гнучка видача автомобіля в прокат (чи потрібно забирати автомобіль за стандартною адресою, чи можна домовитися про зустріч в різних місцях міста);
- color – колір автомобіля;
- characteristics – текст оголошення (опис автомобіля або деяких деталей прокату);
- active – поле, яке визначає, чи оголошення є активним на даний момент, чи воно фактично заархівовано;
- production\_year – рік виробництва автомобіля;
- state – технічний стан автомобіля, яким його визначив власник (обирається категорія стану);
- score – середня оцінка, виставлена автомобілю орендарями за їхніми прокатами.

Поле таблиці `machinemodel_id` є зовнішнім ключем з таблиці `Machinemodel`. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями `Machinemodel` і `Machine`: у програмі може існувати різна кількість автомобілів кожної моделі, проте один автомобіль належить тільки до однієї конкретної моделі автомобіля.

Таким чином автомобіль описується своїми основними властивостями, проте автомобіль можна описати великою кількістю додаткових властивостей, які до того ж можуть мати унікальне значення у конкретного автомобіля. Наприклад, обшивка салону може бути унікальною в автомобілі і відрізнятися від всіх інших автомобілів цієї моделі в принципі. Сутність властивості визначається в БД таблицею `Property` з полями:

- id – ідентифікатор властивості;
- name – назва властивості.

Властивості самі по собі не важливі в розрізі програми, а важливо те, які вони мають значення для конкретного автомобіля. Для внесення цих даних створено таблицю `MachineProperty` з полями:

- id – ідентифікатор властивості автомобіля;

- machine\_id – автомобіль, який характеризується цією властивістю;
- property\_id – властивість, яка характеризує автомобіль;
- property\_value – значення цієї властивості.

Поле таблиці machine\_id є зовнішнім ключем з таблиці Machine. Поле таблиці property\_id є зовнішнім ключем з таблиці Property.

Зв'язок між таблицями Machine і Property має тип багато до багатьох: кожен автомобіль може мати багато властивостей для власного опису, при цьому кожна властивість може використовуватися для опису різних автомобілів. Тому цей зв'язок в реляційній БД реалізовано через створення таблиці MachineProperty.

Сутність клієнта прокату визначається в БД таблицею Client з полями:

- id – ідентифікатор клієнта;
- fullname – повне прізвище та ім'я клієнта;
- username – логін, що використовується клієнтом;
- password – пароль, що використовується клієнтом;
- driverlicense – номер водійського посвідчення клієнта (якщо воно в нього є);
- duedate – дата, до якої дійсне водійське посвідчення;
- passport – серія і номер паспорту клієнта з вказівкою країни;
- score – оцінка, яка була в середньому встановлена клієнту власником автомобілів, які він брав у прокат;
- characteristics – опис клієнта (інформація про себе);
- verified – поле, яке визначає, чи було перевірено документи клієнта верифікованим для цього власником під час видачі автомобіля в прокат.

Сутність власника автомобіля визначається в БД таблицею Owner з полями:

- id – ідентифікатор власника автомобіля;
- fullname – повне прізвище та ім'я власника автомобіля;
- username – логін, що використовується власником автомобіля;
- password – пароль, що використовується власником автомобіля;

- characteristics – опис власника автомобіля (інформація про себе);
- checker – поле, яке визначає, чи може цей користувач виконувати верифікацію документів клієнта, чи ні.

Сутність водія визначається в БД таблицею Driver з полями:

- id – ідентифікатор водія;
- fullname – повне прізвище та ім'я водія;
- username – логін, що використовується водієм;
- password – пароль, що використовується водієм;
- characteristics – опис водія (інформація про себе).

Визначення водіїв, які можуть керувати автомобілем у випадку, якщо клієнт хоче взяти автомобіль на прокат і не вміє керувати автомобілем або не має власного водія, виконується через таблицю DriverMachine з полями:

- id – ідентифікатор запису про керування автомобіля певним водієм у прокаті;
- driver\_id – водій автомобіля;
- machine\_id – автомобіль, який взято в прокат.

Зв'язок між таблицями Driver і Machine має тип багато до багатьох: кожен водій може керувати різними автомобілями, а кожен автомобіль може керувати різними автомобілями. Тому цей зв'язок в реляційній БД реалізовано через створення таблиці DriverMachine.

Сам прокат автомобіля клієнтом визначається в БД таблицею Rental з полями:

- id – ідентифікатор прокату автомобіля;
- machine\_id – автомобіль, який було взято в прокат;
- client\_id – клієнт, який створив цей прокат (початково фактично бронювання прокату);
- state – стан прокату автомобіля;
- created – дата і час створення прокату автомобіля у програмі;
- start\_time – час початку прокату;
- end\_time – час завершення прокату;

- `calculated_price` – обчислена вартість прокату;
- `proposed_price` – запропонована вартість прокату, встановлена власником автомобіля для клієнта;
- `payed_price` – сплачена на поточний момент вартість прокату;
- `deposit` – депозит, який було внесено за прокат автомобіля, якщо його було внесено.

Поле таблиці `machine_id` є зовнішнім ключем з таблиці `Machine`. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями `Machine` і `Rental`: один автомобіль може використовуватися в різних прокатах, проте кожен прокат складається тільки з одного автомобіля (це логічно, оскільки програма направлена на самих власників автомобілів, тобто необхідності створювати певні прокати груп сенсу немає).

Поле таблиці `client_id` є зовнішнім ключем з таблиці `Client`. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями `Client` і `Rental`: один клієнт може створити багато прокатів автомобілів за час роботи в програмі, проте кожен прокат створюється і стосується тільки одного клієнта.

На вартість прокату автомобіля впливає те, які тарифи визначив на прокат свого автомобіля власник. Ці тарифи встановлюються власником за кожним автомобілем окремо і визначаються таблицею `MachinePrice` з полями:

- `id` – ідентифікатор запису про вартість прокату автомобіля;
- `machine_id` – автомобіль, для якого встановлено вартість;
- `start_period` – час початку дії тарифу;
- `end_period` – час закінчення дії тарифу;
- `duration_period` – одиниця часу, за якою встановлено тариф (день, тиждень, місяць);
- `price` – вартість прокату за відповідну одиницю часу.

Автомобіль у прокат може надаватися власником не весь час, а в деякі періоди часу. Для визначення періодів часу, в які автомобіль надається в прокат, створено таблицю `MachineAvailability` з полями:

- `id` – ідентифікатор запису про доступність автомобіля для прокату;
- `machine_id` – автомобіль, щодо якого зроблено запис;
- `start` – початок періоду;
- `end` – закінчення періоду.

Поле таблиці `machine_id` є зовнішнім ключем з таблиці `Machine`. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями `Machine` і `MachineAvailability`: один автомобіль може мати багато записів про його доступність для прокату в певний час, але кожен запис про доступність виконується щодо окремого автомобіля, а не декількох разом.

Власник автомобіля може ідентифікувати дефекти автомобіля, щоб підвищити задоволеність клієнтів від результатів прокату і точніше описати стан автомобіля. Для збереження цих даних використовується таблиця `MachineProblem` з полями:

- `id` – ідентифікатор дефекту автомобіля;
- `machine_id` – автомобіль, до якого відноситься дефект;
- `characteristics` – опис дефекту.

Поле таблиці `machine_id` є зовнішнім ключем з таблиці `Machine`. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями `Machine` і `MachineProblem`: один автомобіль може мати багато дефектів, але кожен дефект є характеристикою окремого автомобіля (одного в кожній ситуації).

Окремо визначена можливість верифікації документів клієнта власником автомобіля. Для цього можуть виділятися конкретні особи, яким це право надається. Або може використовуватися логіка, що такі дії може виконувати будь-хто окрім тих, кому це заборонено. Це фактично додаткова можливість для впевненості власників автомобілів у тому, хто саме

звертається до них з запитом на прокат. Збереження цих даних виконується за допомогою таблиці Verification з полями:

- id – ідентифікатор верифікації;
- client\_id – клієнт, верифікацію документа якого виконано;
- owner\_id – власник автомобіля, який виконав верифікацію;
- created – час, коли було виконано верифікацію;
- document – документ, який було підтверджено;
- characteristics – короткий опис деталей верифікації, якщо це необхідно.

Поле таблиці client\_id є зовнішнім ключем з таблиці Client. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями Client і Verification: кожна верифікація стосується документів деякого клієнта, а верифікації документів можуть виконуватися декілька разів стосовно одного й то ж клієнта (водійське посвідчення, паспорт, інші документи за необхідності, окрім того повторна перевірка іншим власником автомобіля).

Поле таблиці owner\_id є зовнішнім ключем з таблиці Owner. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями Owner і Verification: один власник автомобіля може виконати багато верифікацій документів різних клієнтів, але кожна верифікація виконується тільки одним власником автомобіля.

Для визначення відгуків про автомобілі створено таблицю MachineReview з полями:

- id – ідентифікатор відгуку про автомобіль;
- machine\_id – автомобіль, про який створено відгук;
- client\_id – клієнт, який написав відгук;
- review\_text – текст відгуку;
- score – оцінка відгуку;
- comfort – оцінка комфортності автомобіля;
- technicalstate – оцінка технічного стану автомобіля;

– designstate – оцінка зовнішнього вигляду автомобіля.

Поле таблиці machine\_id є зовнішнім ключем з таблиці Machine. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями Machine і MachineReview: про кожен автомобіль може бути від 0 до великої кількості відгуків, кожен відгук стосується одного автомобіля, про який він і написаний.

Поле таблиці client\_id є зовнішнім ключем з таблиці Client. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями Client і MachineReview: один клієнт може створити багато відгуків про автомобілі, які він брав у прокат, проте кожен такий відгук може мати тільки одного клієнта як автора.

Для визначення відгуків про клієнтів створено таблицю ClientReview з полями:

- id – ідентифікатор відгуку про клієнтів;
- owner\_id – власник автомобіля, який написав відгук;
- client\_id – клієнт, про якого написано відгук;
- review\_text – текст відгуку;
- score – оцінка відгуку.

Поле таблиці owner\_id є зовнішнім ключем з таблиці Owner. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями Owner і ClientReview: один власник автомобіля може створити багато відгуків про клієнтів, кожен відгук про клієнта має тільки одного власника автомобіля, що є автором відгуку.

Поле таблиці client\_id є зовнішнім ключем з таблиці Client. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями Client і ClientReview: про одного клієнта різні власники можуть створити багато відгуків, проте кожен відгук власника автомобіля стосується тільки одного конкретного клієнта.

Оренди можуть відбуватися в різних локаціях, а тому щоб їх ідентифікувати, було виділено сутності міста та країни, що можна

використовувати для зручного пошуку зокрема, адже такий індивідуальний прокат не зручно можна буде виконувати або взагалі неможливо, якщо клієнт і автомобіль знаходяться в різних містах або різних країнах.

Для визначення міст створено таблицю City з полями:

- id – ідентифікатор міста;
- name – назва міста;
- country\_id – країна, де знаходиться місто.

Поле таблиці country\_id є зовнішнім ключем з таблиці Country. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями Country і City: місто знаходиться в конкретній країні, а в країні багато міст.

Для визначення країн створено таблицю Country з полями:

- id – ідентифікатор країни;
- name – назва країни.

Останньою складовою, що відображається у вигляді даних в програмі є повідомлення між власниками автомобілів і клієнтами. Комунікація відбувається саме між ними. Для визначення повідомлень створено таблицю Message з полями:

- id – ідентифікатор повідомлення;
- client\_id – клієнт з повідомлення;
- owner\_id – власник автомобіля з повідомлення;
- created – час створення повідомлення;
- source – поле, яке визначає, чи клієнт створив повідомлення, чи власник автомобіля;
- message\_text – текст повідомлення.

Поле таблиці client\_id є зовнішнім ключем з таблиці Client. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями Client і Message: клієнт може мати багато адресованих йому повідомлень або створених ним, проте кожне повідомлення може мати тільки одного клієнта в якості автора або адресата.

Узагальнена структура БД, що була описана, за допомогою схеми структури БД (рис. 2.4).

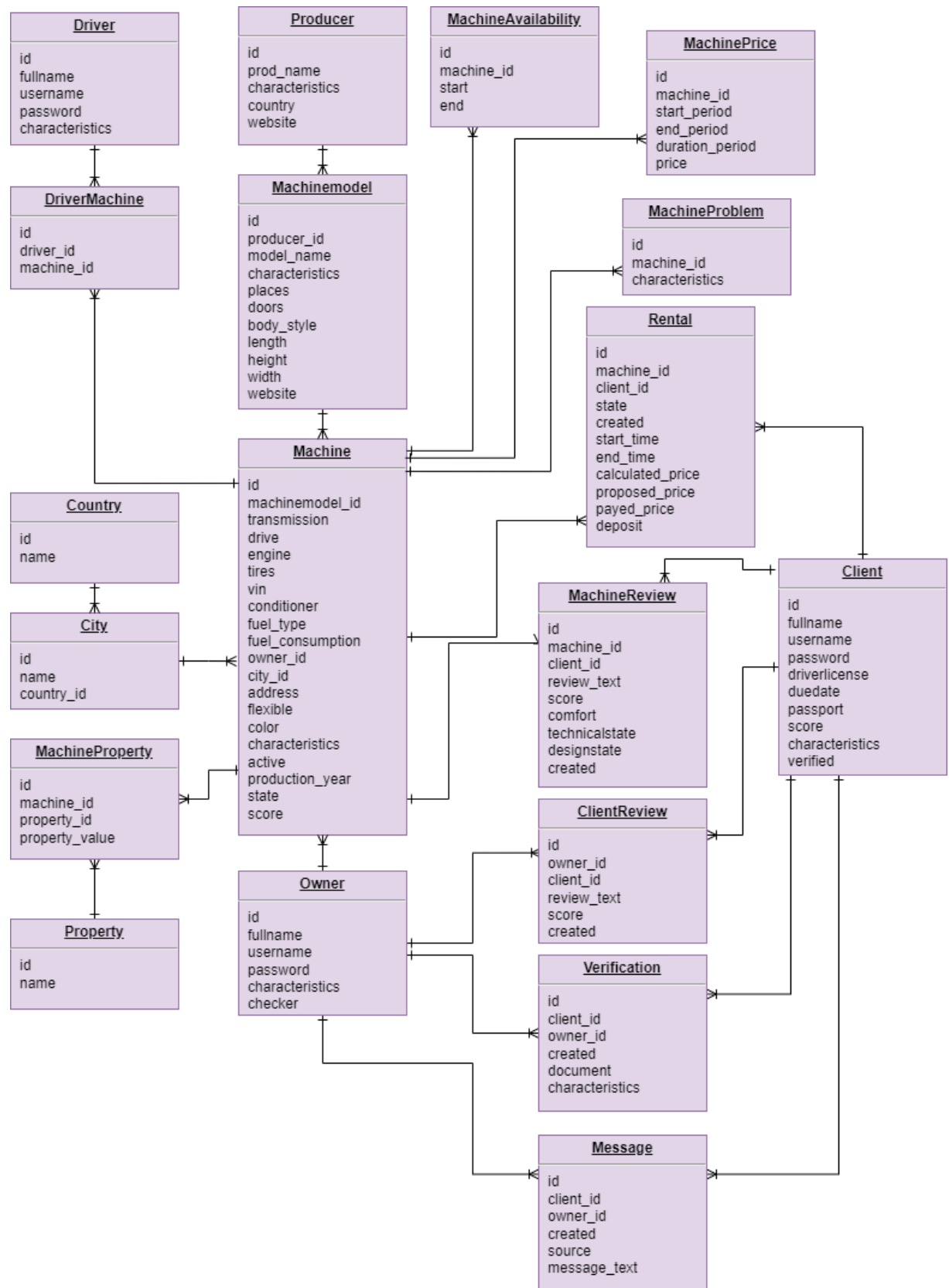


Рисунок 2.4 – Схема структури БД

Поле таблиці `owner_id` є зовнішнім ключем з таблиці `Owner`. За допомогою цього зв'язку визначається відношення типу «один-до-багатьох» між таблицями `Owner` і `Message`: власник автомобіля може мати багато адресованих йому повідомлень або створених ним, проте кожне повідомлення може мати тільки одного власника автомобіля в якості автора або адресата.

## **2.4 Висновки за розділом 2**

У другому розділі порівняно мови програмування під час вибору мови для реалізації ПЗ і обрано мову Python разом з фреймворком Django, а також системою керування БД MySQL для реалізації програми. Виділено функціональні вимоги до програми, побудовано діаграми прецедентів для клієнта, власника автомобіля, водія та адміністратора. Виконано проєктування структури БД для підтримки роботи з функціональністю програми.

## 3 РОЗРОБКА ПРОГРАМИ

### 3.1 Основні принципи організації розробки програми

Зважаючи на використання вебфреймворку Django для розробки, структура застосунку побудована з моделей, представлень і шаблонів. Усі моделі мають структуру у відповідності зі структурою БД, яку було представлено в розділі 2. Для кожної таблиці (окрім таблиць, які реалізують зв'язок «багато-до-багатьох») було створено відповідний клас моделі. Представлення було розроблено у відповідності зі сценаріями роботи користувачів. Принципи організації сценаріїв можна прослідкувати на діаграмі класів (рис. 3.1).

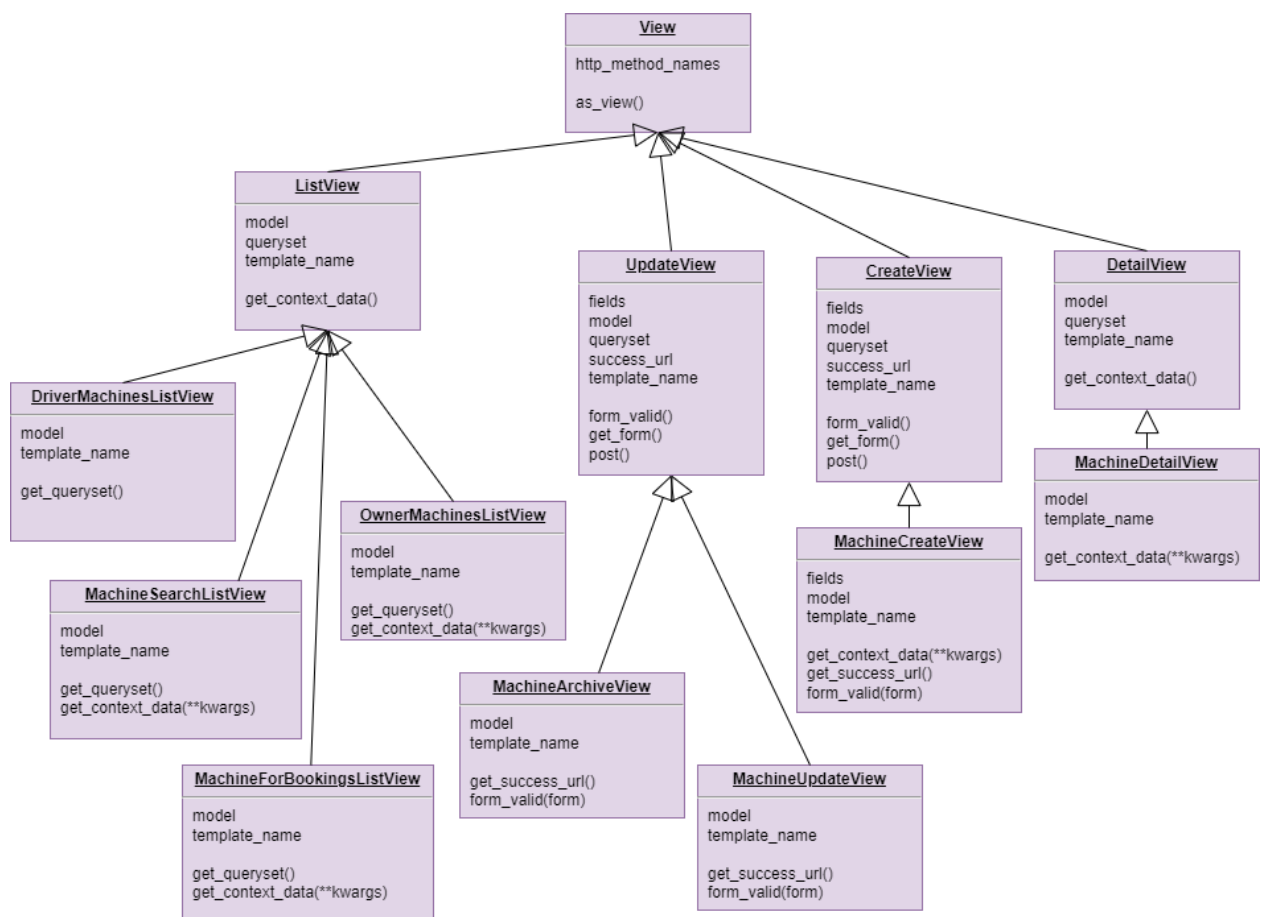


Рисунок 3.1 – Діаграма класів оголошення про прокат автомобіля

На діаграму класів винесено класи, що стосуються оголошення про прокат автомобіля, не включаючи класів, які стосуються дефектів,

доступності автомобіля. Оскільки в основі використано Django, то класи представлень реалізовано через наслідування класів, які існують в Django.

Наслідуванням класу `CreateView` організовано всі класи, які стосуються створення записів (у даному випадку оголошень про прокат автомобілів) у БД.

Наслідування класу `UpdateView` організовано всі класи, які стосуються редагування записів (зокрема оголошень про прокат автомобілів) у БД.

Наслідуванням класу `ListView` організовано роботу з класами, що в основі вимагають відображення переліків (як наприклад, результати пошуку).

Перегляд повних даних запису реалізується наслідуванням класу `DetailView`.

### **3.2 Опис реалізації програми**

У відповідність з конструкцією, описаною на рис. 3.1, було реалізовано класи представлень, що забезпечують наповнення шаблонів відповідних за функціями вебсторінок.

Реалізацію заповнення даними нового оголошення про прокат автомобіля виконано через клас `MachineCreateView`. У ньому через поле `fields` виконується зв'язування полів форми за відповідними полями моделі (зокрема VIN-номер, тип пального, двигун тощо), а частина даних, що стосується моделі автомобіля та міста, де автомобіль знаходиться, вноситься до відповідних полів через оброблення отриманого ідентифікатора і витягання відповідного об'єкта з БД і збереження цього об'єкта у відповідному полі моделі. Через POST-запит і параметр `drivers` передається значення номерів водіїв, які допущені до керування цим автомобілем. Переданий перелік розбирається, виділяються об'єкти, що відповідають кожному водію, а тоді відповідний водій вноситься через зв'язок багато-до-багатьох всередині моделі в полі `drivers` до набору водіїв автомобіля.

Реалізацію заповнення даними оголошення про прокат автомобіля у випадку його редагування виконано через клас `MachineUpdateView`. У ньому через поле `fields` виконується зв'язування полів форми за відповідними полями моделі, завдяки чому автоматично оновлюються змінені значення, номер моделі автомобіля та міста, де автомобіль знаходиться, оновлюється програмно через оброблення отриманого ідентифікатора і витягання відповідного об'єкта з БД і збереження цього об'єкта у відповідному полі моделі. Через POST-запит і параметр `drivers` передається значення номерів водіїв, які допущені до керування цим автомобілем. Оскільки перелік може бути змінено, і водії, які керували автомобілем, можуть бути вилучені, а інші навпаки додані, то спочатку виконується розбір переліку і визначення того, чи є цей водій у полі `drivers`. Якщо водія немає, то він заноситься до переліку. Якщо є, то пропускається. Далі виконується аналіз того, чи всі водії, які занесені в поле `drivers` є в переліку, який був переданий через POST-запит. Якщо виявляється, що деякого автомобіля в полі немає, то він вилучається з БД.

Архівування оголошення реалізується через клас `MachineArchiveView`, у якому змінюється значення поля активності в оголошенні на нульове, що визначає заархівоване оголошення.

Реалізацію заповнення даних, що стосуються конкретного оголошення про прокат автомобіля, виконано через клас `MachineDetailView`, де спочатку витягається інформація про саме оголошення, а далі вона розширяється за рахунок останніх 10 відгуків про цей автомобіль. Усі інші дані, зокрема про дефекти автомобіля, доступність автомобіля, доступні через зворотній зв'язок відповідних полів зовнішніх ключів класів.

Реалізацію пошуку автомобілів для бронювання виконано через створення класу `MachineForBookingsListView`. Основна робота класу реалізована через метод `get_queryset`, в якому дані, що є результатом пошуку відбираються. Розглянемо детальніше реалізацію даного методу.

Спочатку обираються всі оголошення, які є активними на даний момент. Тоді через параметр GET-запиту `producer` виділяється, у випадку, якщо таке значення було задано, номер виробника автомобілів, а за номером встановлюється відповідний об'єкт. Цей об'єкт забезпечує фільтрування всіх активних оголошень за відповідністю цього параметра.

Так покроково продовжується обробка за кожним параметром GET-запиту, що включають зокрема модель автомобіля, кількість місць у автомобілі, кількість дверей у автомобілі, тип кузова автомобіля, тип коробки передач автомобіля, наявність необхідної системи кондиціонування, тип пального, місто, в якому потрібно знайти автомобіль, потреба в тому, щоб розпочати прокат за адресою клієнта, найстаріша дата виробництва та мінімальна оцінка, яку повинен мати в оголошенні автомобіль. Після цього реалізується перевірка того, що на вказані дати доступна оренда відібраного автомобіля. Для цього виконується прохід по всьому переліку автомобілів, які після всієї фільтрації залишились в переліку. Для кожного автомобіля виконується перевірка замовлень прокату, які за ним були здійснені. Перевірка відбувається щодо факту перетину інтервалів часу в замовленні та в заданих параметрах пошуку. Якщо перетин хоч з одним замовленням за часом є, то такий автомобіль вилючається з переліку (або точніше не включається в новий перелік). Якщо перетину немає, то виконується перевірка наявності запису про те, що час, який було зазначено в параметрах пошуку, відповідає одному з інтервалів часу, під час якого доступний автомобіль для прокату, тобто відбувається фільтрація записів з контексту моделі `MachineAvailability`.

Реалізацію внесення значення властивості до оголошення автомобіля виконано через клас `MachinePropertyCreateView`. У ньому через поле `fields` виконується зв'язування поля форми з відповідним полем моделі, що встановлює значення властивості, а номер автомобіля і номер властивості, для якої встановлюється значення, передається через параметри, відповідним

чином обробляється, виділяється об'єкт автомобіля, об'єкт властивості і значення вноситься у поле моделі.

Реалізацію редагування значення властивості до оголошення автомобіля виконано через клас `MachinePropertyUpdateView`. У ньому через поле `fields` виконується зв'язування поля форми з відповідним полем моделі, що встановлює значення властивості у разі оновлення, а номер автомобіля і номер властивості, для якої встановлюється значення, у випадку оновлення передається через параметри, відповідним чином обробляється, виділяється об'єкт автомобіля, об'єкт властивості і значення вноситься у поле моделі.

Реалізацію перегляду відгуків про заданий автомобіль виконано через створення класу `MachineReviewsListView`. Основна робота класу реалізована через метод `get_queryset`, в якому відбираються дані серед всього переліку відгуків про автомобіль за номером автомобіля, який задано в параметрі GET-запиту `machine`, а далі ці відгуки розташовуються за датою створення.

Реалізацію перегляду відгуків, створених про автомобілі клієнтом з заданим номером виконано через створення класу `ClientReviewsListView`. Номер клієнта задано в параметрі GET-запиту `client`.

Реалізацію перегляду відгуків про автомобілі, які були створені клієнтом, який авторизувався в програмі, виконано через створення класу `PersonalReviewsListView`.

Окремо реалізовано клас `MachineSearchListView`, який забезпечує формування переліку даних, що відповідають оголошенням, які хотів знайти клієнт, але які створені без врахування дат бронювання. Тобто це реалізація пошуку на випадок, коли користувач готовий гнучко змінити дати прокату, і йому потрібно знайти автомобілі, які найкраще відповідають його основним вимогам, а потім вже визначити, на які саме дати він хоче створити замовлення прокату.

Реалізацію внесення доступних дат для прокату автомобіля за оголошенням виконано через клас `MachineAvailabilityCreateView`. У ньому через поле `fields` виконується зв'язування полів форми за відповідними

полями моделі (дата, з якої доступний автомобіль у прокаті, та дата, до якої автомобіль доступний у прокаті), а номер автомобіля передається через параметри, відповідним чином обробляється, виділяється об'єкт автомобіля і значення вноситься у поле моделі.

Реалізацію редагування доступного інтервалу дат для прокату автомобіля за оголошенням виконано через клас `MachineAvailabilityUpdateView`. У ньому через поле `fields` виконується зв'язування полів форми на випадок оновлення значень за відповідними полями моделі (дата, з якої доступний автомобіль у прокаті, та дата, до якої автомобіль доступний у прокаті). Номер автомобіля не оновлюється.

Реалізацію перегляду оголошень про прокат автомобілів, які належать власнику, який авторизувався, виконано через створення класу `PersonalMachinesListView`. Основна робота класу реалізована через метод `get_queryset`, в якому спочатку визначають дані авторизації, далі відбирається об'єкт власника автомобіля, який відповідає даним авторизації, відбираються дані серед всього переліку оголошень про прокат автомобілів, за цим власником, а далі оголошення розташовуються, починаючи від останнього створеного.

Реалізацію перегляду оголошень про прокат автомобілів, якими може керувати водій, який авторизувався, виконано через створення класу `DriverMachinesListView`. Основна робота класу реалізована через метод `get_queryset`, в якому спочатку визначають дані авторизації, далі відбирається об'єкт водія, який відповідає даним авторизації, відбираються дані серед всього переліку оголошень про прокат автомобілів, в яких серед переліку водіїв вказано заданого, а далі оголошення розташовуються, починаючи від останнього створеного.

Реалізацію перегляду оголошень про прокат автомобілів, які належать власнику, який авторизувався, виконано через створення класу `OwnerMachinesListView`. Основна робота класу реалізована через метод `get_queryset`, в якому спочатку визначають дані авторизації, далі відбирається

об'єкт власника, який відповідає даним авторизації, відбираються дані серед всього переліку оголошень про прокат автомобілів, в яких власником є заданий, а далі оголошення розташовуються, починаючи від останнього створеного.

За подібною логікою реалізовано інші класи системи, а оскільки ці рішення повторюють опис, який було виконано вже для наявних класів, то вони не представлялись окремо.

### **3.3 Висновки за розділом 3**

Визначено логіку організації коду програми на основі використання вебфреймворку Django з приведенням допису цієї логіки на основі діаграми класів, описано основні розроблені класи програми і принципи їх створення.

## **4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ**

### **4.1 Призначення програми**

Розроблена програма призначена для надання можливості власникам автомобілів виставити на прокат власний автомобіль та впорядковувати такі прокати, а клієнтам відповідно брати автомобілі на прокат у фізичних осіб, а не у компаній, керуючи ними самостійно або з водіями.

### **4.2 Умови виконання програми**

Технічні засоби повинні забезпечувати мінімальні вимоги, що включають наступну мінімальну конфігурацію для практичного використання: 8 Гб оперативної пам'яті для забезпечення роботи сервера, процесор з 4 ядрами в тактову частоту в 2,8 ГГц.

### **4.3 Виконання програми**

Програма запускається на виконання розміщенням файлів проекту на сервері і запуском вебсервера. Після звернення до програми за стандартною адресою відкривається сторінка для роботи над пошуком автомобілів для прокату. Дана сторінка містить пункт меню пошук і кнопку для авторизації для доступу до інших функціональних можливостей. Зі сторінки авторизації можна обрати, яку саме роль користувача необхідно використати для подальшої роботи. В залежності від цього потрібно ввести відповідний логін і пароль, а тоді відкриється панель для роботи відповідного користувача.

Також передбачено відкриття вебзастосунку в режимі адміністратора для внесення даних, зокрема внесення міст, країн, виробників автомобілів та доступних моделей автомобілів. Реалізується стандартно додаванням до вебзастосунку посилання admin в кінці адреси, робота реалізується через табличні форми.

Основною формою взаємодії власника автомобілів з програмою є форма відображення автомобілів (оголошень) власника. Вона відкриється одразу після авторизації власника автомобілів. Кожне оголошення відображається окремо на каруселі зображень, а для переключення між автомобілями використовуються кнопки (рис. 4.1).

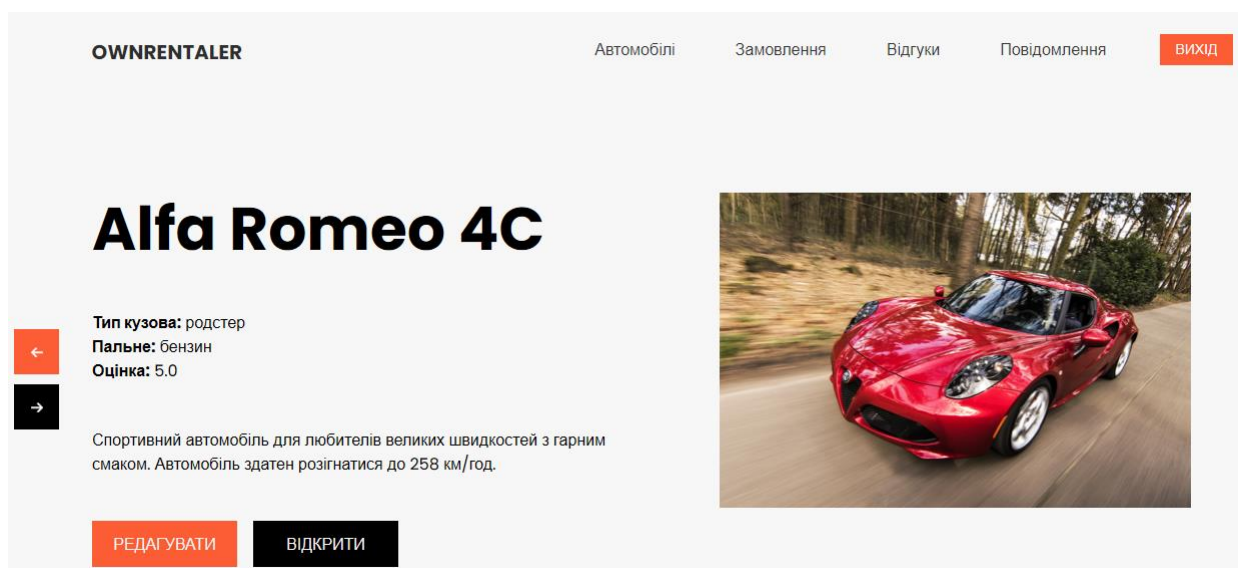


Рисунок 4.1 – Перегляд власних оголошень власником автомобілів

Використовуючи кнопку «Відкрити» під оголошенням, можна відкрити на окремій сторінці всю інформацію про оголошення, що включає повні дані цього оголошення, для власника – перелік водіїв оголошення, перелік властивостей оголошення, перелік дефектів автомобіля, перелік відгуків. При чому на сторінці відображаються тільки перші 10 відгуків, а за допомогою кнопки «Усі» можна переміститися до сторінки відображення абсолютно всіх відгуків про цей автомобіль.

Використовуючи кнопку «Редагувати» можна перейти до внесення відповідних даних, оголошення, при цьому для змінювання переліку властивостей, переліку дефектів відкривається окрема сторінка, а з неї додаються і редагуються необхідні значення.

Цей пункт меню «Автомобілі» відкривається автоматично, але його також можна обрати з меню у будь-який момент

Пункт меню «Замовлення» дозволяє переглянути всі замовлення, обравши відповідний автомобіль, за яким відображаються замовлення. Звідси можна керувати замовлення, зокрема можна редагувати замовлення, скасувати замовлення, підтвердити його та зазначити видачу автомобіля за замовленням і повернення його. Це кнопки, які відображаються біля кожного автомобіля. Усі кнопки окрім редагування залежать від того, в якому саме поточному стані знаходиться замовлення, і в який стан воно може перейти. Таким чином визначається набір кнопок для керування замовленнями. За реалізованими замовленнями доступні кнопки «Відгук», що дозволяє створити відгук, та «Верифікація» для внесення даних про верифіковані документи клієнта. Інформація про верифікацію клієнта потім доступна з його сторінки, що відкривається при натисканні на логін клієнта.

Пункт меню «Відгуки» дозволяє переглянути всі відгуки, керуючи ними за допомогою кнопок «Власні» і «Автомобілі». Перший варіант відображає власні відгуки, створені власником автомобіля, а другий варіант дозволяє переглянути відгуки за обраним з переліку власним автомобілем.

Пункт меню «Повідомлення» дозволяє переглянути повідомлення, які були направлені власнику автомобіля і відправлені ним відповідно за клієнтами.

У свою чергу клієнт має окреме меню власного інтерфейсу, що складається з наступних пунктів.

Пункт меню «Пошук» (рис. 4.2) дозволяє виконати пошук автомобілів загалом для прокату або для прокату в задані дати. Під час пошуку можна за допомогою фільтрів вказати назву виробника автомобіля, назву моделі, кількість місць у автомобілі, кількість дверей, тип кузова, тип коробки передач, тип системи кондиціонування, тип пального, місто, де знаходиться автомобіль, найстарший рік виробництва, мінімальний середній бал та необхідність зустрічі для прокату на власній або нейтральній території. У випадку натискання кнопки «Прокат» вгорі сторінки потрібно буде також задати інтервал дат. У випадку натискання кнопки «Загалом» вгорі сторінки

будуть відображатися всі знайдені активні оголошення без врахування їх доступності на вказані дати.

OWNRENTALER
Пошук
Замовлення
Повідомлення
Вихід

## Результати пошуку

До пошуку

### Lamborghini Aventador

Власник: gabriel  
Пальне: бензин  
Рік виробництва: 2021  
Вартість: 7000 грн./день  
Оцінка: 5.0

Відкрити

Замовити



### Lamborghini Gallardo

Власник: illya  
Пальне: бензин  
Рік виробництва: 2020  
Вартість: 6500 грн./день  
Оцінка: 5.0

Відкрити

Замовити



Рисунок 4.2 – Результати пошуку автомобілів для прокату

Результати пошуку відкриваються на окремій сторінці, а повернутися до визначення параметрів пошуку можна за допомогою кнопки «До пошуку». Кожен результат пошуку відображається у окремому рядку. За кожним оголошенням виводиться коротка інформація, що включає назву моделі автомобіля з назвою виробника, логін власника, тип пального, рік виробництва, фотографію автомобіля, вартість прокату на день (уточнити на триваліший строк вартість можна вже під час формування замовлення). Для виконання дій над оголошеннями надаються кнопки «Відкрити» для

відкриття сторінки з повною інформацією оголошення та «Замовити» для переходу до оформлення замовлення даного автомобіля (під час оформлення замовлення потрібно вказати строки прокату).

При відкритті сторінки оголошення можна отримати повну інформацію про автомобіль, прокат якого пропонується, можна перейти до сторінки власника. Окрім того внизу сторінки знаходиться перелік відгуків про автомобіль з можливістю відкриття всіх відгуків (рис. 4.3).

## Відгуки про автомобіль



sergey

Автомобіль дуже сподобався. Виявився і комфортабельним, і зовнішній вигляд не давав спокою всій сім'ї, ще й стан автомобіля прекрасний. Дякую.

**Оцінка: 5**

**Комфорт: 5**

**Технічний стан: 5**

**Дизайн: 5**



Рисунок 4.3 – Відгуки за автомобілем

На самій сторінці відображається кожен відгук окремо (рис. 4.3), а перемикатися між ними можна за допомогою кнопок вліво та вправо внизу сторінки.

Пункт меню «Замовлення» дозволяє переглянути перелік всіх власних замовлень авторизованого клієнта. За необхідності замовлення можна скасувати. За реалізованими замовленнями доступна кнопка «Відгук», що дозволяє створити відгук.

Пункт меню «Повідомлення» дозволяє переглядати і відправляти повідомлення власникам автомобілів.

#### **4.4 Тестування програми**

Тестування програми виконувалось перевіркою функцій програми, які були визначені для реалізації на основі ітеративного процесу, під час якого спочатку були виявлені наявні проблеми, потім вони були вирішені, а після цього виконувались повторні перевірки. У підсумку було успішно перевірено виконання всіх функцій програми.

#### **4.5 Висновки за розділом 4**

У розділі представлено опис тестування програми, яке було виконано і успішно завершено, надано опис роботи користувачів з програмою з визначенням порядку цієї роботи, що ґрунтується на пунктах меню та переходах за посиланнями між сторінками.

## ВИСНОВКИ

У дипломній роботі необхідно було розробити програмне забезпечення надання власних автомобілів у прокат для підтримки взаємодії між власниками автомобілів і орендарями. Для цього було визначено набір завдань, які було виконано в роботі.

За результатами огляду проблеми прокату автомобілів загалом було визначено підпроблему прокату автомобілів, що надаються їх власниками. Цей варіант прокату розглянуто як дворанговий спосіб і порівняно з одноранговим. Проаналізовано програми, які для такого дворангового прокату можуть застосовуватися: Turo, Rentben. Виділено особливості програми, яка розробляється, порівняно її з цими аналогами.

Порівняно мови програмування під час вибору мови для реалізації ПЗ і обрано мову Python разом з фреймворком Django, а також системою керування БД MySQL для реалізації програми. Виділено функціональні вимоги до програми, побудовано діаграми прецедентів для клієнта, власника автомобіля, водія. Виконано проєктування структури БД для підтримки роботи з функціональністю програми.

Визначено логіку організації коду програми на основі використання вебфреймворку Django, описано основні розроблені класи програми і принципи їх створення.

Розроблено вебзастосунок для надання можливості власникам автомобілів виставити на прокат власний автомобіль та впорядковувати такі прокати, а клієнтам відповідно брати автомобілі на прокат у фізичних осіб, а не у компаній, керуючи ними самостійно або з водіями. ПЗ таким чином дозволяє впорядкувати сферу індивідуальних прокатів і створити можливість тимчасового заробітку власника автомобілів у час, коли автомобіль простоює і не використовується.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Klochikhin, E. Car sharing hurdles [Electronic resource] : What can boost the economy of shared mobility? / E. Klochikhin. – Access mode : <https://medium.com/predict/car-sharing-hurdles-what-can-boost-the-economy-of-shared-mobility-a789c803c5fa>.
2. Zeng, H. What's Stopping Carsharing? 4 Challenges Holding Emerging Markets Back [Electronic resource] / H. Zeng. – Access mode : <https://thecityfix.com/blog/stopping-carsharing-report-4-challenges-holding-emerging-markets-back-heshuang-zeng/>.
3. Laya, A. Shared mobility [Electronic resource] : Why 'they' should all be sharing their cars / A. Laya. – Access mode : <https://www.ericsson.com/en/blog/2021/3/shared-mobility-they-should-share-cars>.
4. Hall-Geisler, K. The Trouble With Car Sharing [Electronic resource] / K. Hall-Geisler. – Access mode : <https://auto.howstuffworks.com/tech-transport/car-sharing-trouble.htm>.
5. Helling, B. Peer-to-Peer Car Rental [Electronic resource] : What Are Your Options? / B. Helling. – Access mode : <https://www.ridester.com/peer-to-peer-car-rental/>.
6. HyreCar Vs. Turo [Electronic resource] : Best Platform to Rent Your Car? – Access mode : <https://www.hyrecar.com/blog/best-car-rental-platform/>.
7. Turo car sharing marketplace. Rent the perfect car [Electronic resource]. – Access mode : <https://turo.com/>.
8. RentBen – Rental. Car share [Electronic resource]. – Access mode : <https://rentben.com/>.
9. Ramalho, L. Fluent Python [Text] : Clear, Concise, and Effective Programming / L. Ramalho. – Massachusetts : O'Reilly Media, 2022. – 1012 p.
10. Shaw, Z. Learn Python 3 the Hard Way [Text] : A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code / Z. Shaw. – Boston, Massachusetts : Addison-Wesley Professional, 2017. – 320 p.

11. Percival, H. Test-Driven Development with Python [Text] : Obey the Testing Goat: Using Django, Selenium, and JavaScript / H. Percival. – Massachusetts : O'Reilly Media, 2017. – 624 p.

**ДОДАТОК А**  
**Технічне завдання**

## **Вступ**

Послуги прокату автомобілів є достатньо затребуваними, але надаються вони централізованими компаніями, у той же час є простір для розширення прокату за рахунок індивідуальних власників. Вони можуть надавати свої автомобілі за оголошеннями, знайомим. Тому важливо створити програмне забезпечення для підтримки саме індивідуальних власників автомобілів.

### **A.1 Підстави для розробки**

Розробка в межах дипломної кваліфікаційної роботи реалізується на підставі наказу № 102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка» за темою «Програмне забезпечення надання власних автомобілів у прокат».

### **A.2 Призначення розробки**

Програма має бути в результаті призначена для надання можливості власникам автомобілів виставити на прокат власний автомобіль та впорядковувати такі прокати, а клієнтам відповідно брати автомобілі на прокат у фізичних осіб, а не у компаній, керуючи ними самостійно або з водіями.

### **A.3 Основні вимоги до програми**

#### **A.3.1 Вимоги до функціональних характеристик**

Функціональні характеристики, що має охоплювати програма, повинні включати:

- створення оголошення про прокат власного автомобіля;

- замовлення прокату автомобіля;
- редагування замовлення прокату автомобіля;
- підтвердження замовлення;
- скасування замовлення;
- керування станом замовлення;
- створення відгуку про автомобіль клієнтом;
- створення відгуку про клієнта власником автомобіля;
- перегляд відгуків про клієнта;
- перегляд відгуків про автомобіль;
- пошук автомобіля для прокату;
- перегляд замовлень прокату автомобіля власником;
- додавання водіїв для керування автомобілем;
- вилучення водіїв від керування автомобілем;
- перегляд автомобілів, якими може керувати водій;
- перегляд замовлень прокатів, якими може керувати водій;
- авторизація, реєстрація користувачів;
- додавання дефектів автомобіля;
- редагування дефектів автомобіля;
- визначення вартості прокату автомобіля на період часу;
- змінювання вартості прокату автомобіля;
- визначення періоду доступності автомобіля для прокату;
- верифікація документів клієнта;
- перегляд оголошень власника;
- архівування оголошення.

### **A.3.2 Умови експлуатації**

Для підтримки експлуатації програми потрібно розгорнути на вебсервері застосунок, відповідно потрібен щонайменше один адміністратор, який зможе керувати роботою сервера БД, вебсервера та підтримкою запуску

вебзастосунку спершу і подальшою підтримкою щодо роботи з даними (внесення даних).

### **А.3.3 Вимоги до складу та параметрів технічних засобів**

Технічні засоби повинні забезпечувати мінімальні вимоги, що включають наступну мінімальну конфігурацію для практичного використання: 8 Гб оперативної пам'яті для забезпечення роботи сервера, процесор з 4 ядрами в тактову частоту в 2,8 ГГц.

### **А.4 Порядок контролю та приймання**

Виконання дипломної роботи реалізується у відповідності з календарним планом, приймання роботи відбувається під час процедури захисту дипломної роботи.

**ДОДАТОК Б**  
**Текст програми**

```

from django.views.generic.edit import CreateView, DeleteView
from django.views.generic import ListView, UpdateView
from django.views.generic.detail import DetailView
from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from datetime import date, datetime
from .models import Machine, MachinePrice, MachineAvailability,
MachineProblem, Producer, Machinemodel, Property, MachineProperty,
Driver, Owner, Rental

```

```

class MachineCreateView (LoginRequiredMixin, CreateView):
    model = Machine
    fields = ['transmission',
              'drive',
              'engine',
              'tires',
              'vin',
              'conditioner',
              'fuel_type',
              'fuel_consumption',
              'address',
              'flexible',
              'color',
              'characteristics',
              'active',
              'production_year',
              'state']
    template_name = 'add_machine_advertisement.html'

    def get_success_url(self):
        return reverse_lazy('machineview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        machine_model = get_object_or_404(Machinemodel,
id=self.kwargs['machinemodel'])
        kwargs['machinemodel'] = machine_model
        city = get_object_or_404(City, id=self.kwargs['city'])

```

```

        kwargs['city'] = city
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        machine_model = get_object_or_404(Machinemodel,
id=self.kwargs['machinemodel'])
        self.machinemodel_id = machine_model
        city = get_object_or_404(Machinemodel,
id=self.kwargs['city'])
        self.city_id = city
        user_authorized = self.request.user
        owner = Owner.object.get(id = user_authorized.id)
        self.owner_id = owner
        self.score = 0
        machine_drivers =
str(self.request.POST.get('drivers'))
        machine_drivers_list = machine_drivers.split(',')
        for driver in machine_drivers_list:
            driver = Driver.objects.get(id = driver)
            self.drivers.add(driver)
        return super().form_valid(form)

class MachineUpdateView (LoginRequiredMixin, UpdateView):
    model = Machine
    fields = ['transmission',
        'drive',
        'engine',
        'tires',
        'vin',
        'conditioner',
        'fuel_type',
        'fuel_consumption',
        'address',
        'flexible',
        'color',
        'characteristics',
        'active',
        'production_year',
        'state']

```

```

template_name = 'update_machine_advertisement.html'

def get_success_url(self):
    return reverse_lazy('machineview', kwargs={'pk':
self.object.pk})

def get_context_data(self, **kwargs):
    machine_model = get_object_or_404(Machinemodel,
id=self.kwargs['machinemodel'])
    kwargs['machinemodel'] = machine_model
    city = get_object_or_404(City, id=self.kwargs['city'])
    kwargs['city'] = city
    cities = City.objects.all()
    kwargs['cities'] = cities
    producers = Producer.objects.all()
    kwargs['producers'] = producers
    models = Machinemodel.objects.all()
    kwargs['machinemodels'] = models
    return super().get_context_data(**kwargs)

def form_valid(self, form):
    machine_model = get_object_or_404(Machinemodel,
id=self.kwargs['machinemodel'])
    self.machinemodel_id = machine_model
    city = get_object_or_404(Machinemodel,
id=self.kwargs['city'])
    self.city_id = city
    machine_drivers =
str(self.request.POST.get('drivers'))
    machine_drivers_list = machine_drivers.split(',')
    driver_list = []
    for driver in machine_drivers_list:
        driver = Driver.objects.get(id = driver)
        if driver not in self.drivers:
            self.drivers.add(driver)
            driver_list.append(driver)
    for driver in self.drivers:
        if driver not in driver_list:
            self.drivers.remove(driver)

```

```

        return super().form_valid(form)

class MachinePropertyDeleteView (DeleteView):
    model = MachineProperty

class DriverMachineDeleteView (DeleteView):
    model = DriverMachine

class MachinePriceDeleteView (DeleteView):
    model = MachinePrice

class MachineAvailabilityDeleteView (DeleteView):
    model = MachineAvailability

class MachineProblemDeleteView (DeleteView):
    model = MachineProblem

class MachineArchiveView (LoginRequiredMixin, UpdateView):
    model = Machine
    template_name = 'archive_machine_advertisement.html'

    def get_success_url(self):
        return reverse_lazy('machineview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        self.active = 0
        return super().form_valid(form)

class MachineDetailView (DetailView):
    model = Machine
    template_name = 'machine_advertisement.html'

    def get_context_data(self, **kwargs):
        machine_with_reviews =
super().get_context_data(**kwargs)
        reviews = MachineReview.objects.filter(machine_id =
self.object.pk)
        reviews = reviews.order_by('-created')

```

```

        machine_with_reviews ['reviews'] = reviews[:10]
        return machine_with_reviews

class PersonalMachinesListView (ListView):
    model = Machine
    template_name = 'personal_machines.html'

    def get_queryset(self):
        user_authorized = self.request.user
        owner = Owner.object.get(id = user_authorized.id)
        owner_machines = Machine.objects.filter(owner_id =
owner)

        owner_machines = owner_machines.order_by('-id')
        return owner_machines

class DriverMachinesListView (ListView):
    model = Machine
    template_name = 'driver_machines.html'

    def get_queryset(self):
        user_authorized = self.request.user
        driver = Driver.object.get(id = user_authorized.id)
        driver_machines =
Machine.objects.filter(drivers__contains = driver)
        driver_machines = owner_machines.order_by('-id')
        return driver_machines

class OwnerMachinesListView (ListView):
    model = Machine
    template_name = 'personal_machines.html'

    def get_queryset(self):
        get_owner = self.request.GET.get('owner')
        owner = Owner.object.get(id = get_owner)
        owner_machines = Machine.objects.filter(owner_id =
owner)

        owner_machines = owner_machines.order_by('-id')
        return owner_machines

```

```

    def get_context_data(self, **kwargs):
        result_context = super(OwnerMachinesListView,
self).get_context_data(**kwargs)
        get_owner = self.request.GET.get('owner')
        result_context ['owner'] = get_owner
        return result_context

class ProducerCreateView (LoginRequiredMixin, CreateView):
    model = Producer
    fields = ['prod_name',
              'characteristics',
              'website']
    template_name = 'add_producer.html'

    def get_success_url(self):
        return reverse_lazy('producerview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        country = get_object_or_404(Country,
id=self.kwargs['country'])
        kwargs['country'] = country
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        country = get_object_or_404(Country,
id=self.kwargs['country'])
        self.country_id = country
        return super().form_valid(form)

class ProducerUpdateView (LoginRequiredMixin, UpdateView):
    model = Producer
    fields = ['prod_name',
              'characteristics',
              'website']
    template_name = 'update_producer.html'

    def get_success_url(self):

```

```

        return reverse_lazy('producerview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        country = get_object_or_404(Country,
id=self.kwargs['country'])
        kwargs['country'] = country
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        country = get_object_or_404(Country,
id=self.kwargs['country'])
        self.country_id = country
        return super().form_valid(form)

class ProducersListView (ListView):
    model = Machine
    template_name = 'admin_producers.html'

class MachinemodelCreateView (LoginRequiredMixin, CreateView):
    model = Machinemodel
    fields = ['model_name',
        'characteristics',
        'places',
        'doors',
        'body_style',
        'length',
        'height',
        'width',
        'website']
    template_name = 'add_machinemodel.html'

    def get_success_url(self):
        return reverse_lazy('machinemodelview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        producer = get_object_or_404(Producer,
id=self.kwargs['producer'])

```

```

        kwargs['producer'] = producer
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        producer = get_object_or_404(Producer,
id=self.kwargs['producer'])
        self.producer_id = producer
        return super().form_valid(form)

class MachinemodelUpdateView (LoginRequiredMixin, UpdateView):
    model = Machinemodel
    fields = ['model_name',
        'characteristics',
        'places',
        'doors',
        'body_style',
        'length',
        'height',
        'width',
        'website']
    template_name = 'update_machinemodel.html'

    def get_success_url(self):
        return reverse_lazy('machinemodelview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        producer = get_object_or_404(Producer,
id=self.kwargs['producer'])
        kwargs['producer'] = producer
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        producer = get_object_or_404(Producer,
id=self.kwargs['producer'])
        self.producer_id = producer
        return super().form_valid(form)

class MachinePropertiesListView (ListView):

```

```

model = MachineProperty
template_name = 'machine_properties.html'

def get_queryset(self):
    get_machine = self.request.GET.get('machine')
    machine = Machine.object.get(id = get_machine)
    properties = MachineProperty.objects.filter(machine_id
= machine)

    properties = properties.order_by('property_id__name')
    return properties

def get_context_data(self, **kwargs):
    result_context = super(MachinePropertiesListView,
self).get_context_data(**kwargs)
    get_machine = self.request.GET.get('machine')
    result_context ['machine'] = get_machine
    return result_context

class MachinePropertyCreateView (LoginRequiredMixin,
CreateView):
    model = MachineProperty
    fields = ['property_value']
    template_name = 'add_machine_property.html'

    def get_success_url(self):
        return reverse_lazy('machinepropertiesview',
kwargs={'pk': self.object.machine_id.pk})

    def get_context_data(self, **kwargs):
        property = get_object_or_404(Property,
id=self.kwargs['property'])
        kwargs['property'] = property
        machine = get_object_or_404(Machine,
id=self.kwargs['machine'])
        kwargs['machine'] = machine
        return super().get_context_data(**kwargs)

    def form_valid(self, form):

```

```

        property = get_object_or_404(Property,
id=self.kwargs['property'])
        machine = get_object_or_404(Machine,
id=self.kwargs['machine'])
        self.machine_id = machine
        self.property_id = property
        return super().form_valid(form)

```

```

class MachinePropertyUpdateView (LoginRequiredMixin,
UpdateView):

```

```

    model = MachineProperty
    fields = ['property_value']
    template_name = 'update_machine_property.html'

```

```

    def get_success_url(self):
        return reverse_lazy('machinepropertiesview',
kwargs={'pk': self.object.machine_id.pk})

```

```

class MachinePricesListView (ListView):

```

```

    model = MachinePrice
    template_name = 'machine_prices.html'

```

```

    def get_queryset(self):
        get_machine = self.request.GET.get('machine')
        machine = Machine.objects.get(id = get_machine)
        prices = MachinePrice.objects.filter(machine_id =
machine)

        moment_from = date.today()
        prices = prices.filter(start_period__gte =
moment_from)

        prices = prices.order_by('start_period')
        return prices

```

```

    def get_context_data(self, **kwargs):
        result_context = super(MachinePricesListView,
self).get_context_data(**kwargs)
        get_machine = self.request.GET.get('machine')
        result_context ['machine'] = get_machine
        return result_context

```

```

class MachinePriceCreateView (LoginRequiredMixin, CreateView):
    model = MachinePrice
    fields = ['start_period',
              'end_period',
              'duration_period',
              'price']
    template_name = 'add_machine_price.html'

    def get_success_url(self):
        return reverse_lazy('machinepropertiesview',
kwargs={'pk': self.object.machine_id.pk})

    def get_context_data(self, **kwargs):
        machine = get_object_or_404(Machine,
id=self.kwargs['machine'])
        kwargs['machine'] = machine
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        machine = get_object_or_404(Machine,
id=self.kwargs['machine'])
        self.machine_id = machine
        return super().form_valid(form)

class MachinePropertyUpdateView (LoginRequiredMixin,
UpdateView):
    model = MachineProperty
    fields = ['start_period',
              'end_period',
              'duration_period',
              'price']
    template_name = 'update_machine_price.html'

    def get_success_url(self):
        return reverse_lazy('machinepropertiesview',
kwargs={'pk': self.object.machine_id.pk})

class MachineForBookingsListView (ListView):
    model = Machine

```

```

template_name = 'machine_search_for_booking.html'

def get_queryset(self):
    machines = Machine.objects.filter(active = 1)
    producer = self.request.GET.get('producer')
    if producer:
        producer = Producer.objects.get(id = producer)
        machines =
machines.filter(machinemodel_id__producer_id = producer)
        machinemodel = self.request.GET.get('machinemodel')
        if machinemodel:
            machmodel = Machinemodel.objects.get(id =
machinemodel)
            machines = machines.filter(machinemodel_id =
machmodel)

        places = self.request.GET.get('places')
        if places:
            machines =
machines.filter(machinemodel_id__places = places)
        doors = self.request.GET.get('doors')
        if doors:
            machines = machines.filter(machinemodel_id__doors
= doors)

        body_style = self.request.GET.get('body_style')
        if body_style:
            machines =
machines.filter(machinemodel_id__body_style = body_style)
        transmission = self.request.GET.get('transmission')
        if transmission:
            machines = machines.filter(transmission =
transmission)

        conditioner = self.request.GET.get('conditioner')
        if conditioner:
            machines = machines.filter(conditioner =
conditioner)

        fuel_type = self.request.GET.get('fuel_type')
        if fuel_type:
            machines = machines.filter(fuel_type = fuel_type)
        city = self.request.GET.get('city')

```

```

        if city:
            city = City.objects.get(id = city)
            machines =
machines.filter(machinemodel_id__body_style = body_style)
            flexible = self.request.GET.get('flexible')
            if flexible:
                machines = machines.filter(flexible = 1)
            production = self.request.GET.get('production')
            if production:
                machines = machines.filter(production_year__gte =
production)

            score = self.request.GET.get('score')
            if score:
                machines = machines.filter(score__gte =
production)

            rental_start = self.request.GET.get('date1')
            rental_start = datetime.strptime(rental_start,
"%d.%m.%Y")

            rental_end = self.request.GET.get('date2')
            rental_end = datetime.strptime(rental_end, "%d.%m.%Y")
            machine_list = []
            for machine in machines:
                rentals1 = Rental.objects.filter(machine_id =
machine, start_time__lte = rental_start, end_time__gte = rental_end)
                rentals2 = Rental.objects.filter(machine_id =
machine, start_time__gte = rental_start, end_time__lte = rental_end)
                rentals3 = Rental.objects.filter(machine_id =
machine, start_time__gte = rental_start, end_time__lte = rental_end,
start_time__lte = rental_end)
                rentals4 = Rental.objects.filter(machine_id =
machine, start_time__gte = rental_start, end_time__gte = rental_end, ,
start_time__lte = rental_end)
                if not rentals1 and not rentals2 and not rentals3
and not rentals4:
                    availability =
MachineAvailability.objects.filter(machine_id = machine, start__lte =
rental_start, end__gte = rental_end)
                    if availability:
                        machine_list.append(machine)

```

```

        machine_list = machine_list.order_by('-score')
        return machine_list

    def get_context_data(self, **kwargs):
        result_context = super(MachineForBookingsListView,
self).get_context_data(**kwargs)
        producer = self.request.GET.get('producer')
        result_context ['producer'] = producer
        result_context ['producers'] = Producer.objects.all()
        machinemodel = self.request.GET.get('machinemodel')
        result_context ['machinemodel'] = machinemodel
        result_context ['machinemodels'] =
Machinemodel.objects.all()
        places = self.request.GET.get('places')
        result_context ['places'] = places
        doors = self.request.GET.get('doors')
        result_context ['doors'] = doors
        body_style = self.request.GET.get('body_style')
        result_context ['body_style'] = body_style
        transmission = self.request.GET.get('transmission')
        result_context ['transmission'] = transmission
        conditioner = self.request.GET.get('conditioner')
        result_context ['conditioner'] = conditioner
        fuel_type = self.request.GET.get('fuel_type')
        result_context ['fuel_type'] = fuel_type
        city = self.request.GET.get('city')
        result_context ['city'] = city
        result_context ['cities'] = City.objects.all()
        result_context ['countries'] = Country.objects.all()
        flexible = self.request.GET.get('flexible')
        result_context ['flexible'] = flexible
        production = self.request.GET.get('production')
        result_context ['production'] = production
        score = self.request.GET.get('score')
        result_context ['score'] = score
        return result_context

class MachineAvailabilitiesListView (ListView):
    model = MachineAvailability

```

```

template_name = 'machine_availability.html'

def get_queryset(self):
    get_machine = self.request.GET.get('machine')
    machine = Machine.object.get(id = get_machine)
    availabilities =
MachineAvailability.objects.filter(machine_id = machine)
    moment_from = date.today()
    availabilities =
availabilities.filter(start_period__gte = moment_from)
    availabilities = availabilities.order_by('start')
    return availabilities

def get_context_data(self, **kwargs):
    result_context = super(MachineAvailabilitiesListView,
self).get_context_data(**kwargs)
    get_machine = self.request.GET.get('machine')
    result_context ['machine'] = get_machine
    return result_context

class MachineAvailabilityCreateView (LoginRequiredMixin,
CreateView):
    model = MachineAvailability
    fields = ['start',
        'end']
    template_name = 'add_machine_availability.html'

    def get_success_url(self):
        return reverse_lazy('machineavailabilitiesview',
kwargs={'pk': self.object.machine_id.pk})

    def get_context_data(self, **kwargs):
        machine = get_object_or_404(Machine,
id=self.kwargs['machine'])
        kwargs['machine'] = machine
        return super().get_context_data(**kwargs)

    def form_valid(self, form):

```

```

        machine = get_object_or_404(Machine,
id=self.kwargs['machine'])
        self.machine_id = machine
        return super().form_valid(form)

class MachineAvailabilityUpdateView (LoginRequiredMixin,
UpdateView):
    model = MachineAvailability
    fields = ['start',
        'end']
    template_name = 'update_machine_availability.html'

    def get_success_url(self):
        return reverse_lazy('machineavailabilitiesview',
kwargs={'pk': self.object.machine_id.pk})

class MachineProblemsListView (ListView):
    model = MachineProblem
    template_name = 'machine_problems.html'

    def get_queryset(self):
        get_machine = self.request.GET.get('machine')
        machine = Machine.object.get(id = get_machine)
        problems = MachineProblem.objects.filter(machine_id =
machine)

        problems = problems.order_by('id')
        return problems

    def get_context_data(self, **kwargs):
        result_context = super(MachineProblemsListView,
self).get_context_data(**kwargs)
        get_machine = self.request.GET.get('machine')
        result_context ['machine'] = get_machine
        return result_context

class MachineProblemCreateView (LoginRequiredMixin, CreateView):
    model = MachineProblem
    fields = ['characteristics']
    template_name = 'add_machine_problem.html'

```

```

    def get_success_url(self):
        return reverse_lazy('machineproblemsview',
                               kwargs={'pk': self.object.machine_id.pk})

    def get_context_data(self, **kwargs):
        machine = get_object_or_404(Machine,
                                     id=self.kwargs['machine'])
        kwargs['machine'] = machine
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        machine = get_object_or_404(Machine,
                                     id=self.kwargs['machine'])
        self.machine_id = machine
        return super().form_valid(form)

class MachineProblemUpdateView (LoginRequiredMixin, UpdateView):
    model = MachineProblem
    fields = ['characteristics']
    template_name = 'update_machine_problem.html'

    def get_success_url(self):
        return reverse_lazy('machineproblemsview',
                               kwargs={'pk': self.object.machine_id.pk})

class MachineReviewsListView (ListView):
    model = MachineReview
    template_name = 'machine_reviews.html'

    def get_queryset(self):
        get_machine = self.request.GET.get('machine')
        machine = Machine.objects.get(id = get_machine)
        reviews = MachineReview.objects.filter(machine_id =
machine)

        reviews = reviews.order_by('-created')
        return reviews

    def get_context_data(self, **kwargs):

```

```

        result_context = super(MachineProblemsListView,
self).get_context_data(**kwargs)
        get_machine = self.request.GET.get('machine')
        result_context ['machine'] = get_machine
        return result_context

```

```

class ClientReviewsListView (ListView):
    model = MachineReview
    template_name = 'machine_reviews_for_client.html'

    def get_queryset(self):
        get_client = self.request.GET.get('client')
        client = Client.object.get(id = get_client)
        reviews = MachineReview.objects.filter(client_id =
client)

        reviews = reviews.order_by('-created')
        return reviews

    def get_context_data(self, **kwargs):
        result_context = super(ClientReviewsListView,
self).get_context_data(**kwargs)
        get_client = self.request.GET.get('client')
        result_context ['client'] = get_client
        return result_context

```

```

class PersonalReviewsListView (ListView):
    model = MachineReview
    template_name = 'machine_reviews_auth.html'

    def get_queryset(self):
        user_authorized = self.request.user
        client = Client.object.get(id = user_authorized.id)
        reviews = MachineReview.objects.filter(client_id =
client)

        reviews = reviews.order_by('-created')
        return reviews

```

```

class MachineSearchListView (ListView):
    model = Machine

```

```

template_name = 'machine_search.html'

def get_queryset(self):
    machines = Machine.objects.filter(active = 1)
    producer = self.request.GET.get('producer')
    if producer:
        producer = Producer.objects.get(id = producer)
        machines =
machines.filter(machinemodel_id__producer_id = producer)
        machinemodel = self.request.GET.get('machinemodel')
        if machinemodel:
            machmodel = Machinemodel.objects.get(id =
machinemodel)
            machines = machines.filter(machinemodel_id =
machmodel)

        places = self.request.GET.get('places')
        if places:
            machines =
machines.filter(machinemodel_id__places = places)
        doors = self.request.GET.get('doors')
        if doors:
            machines = machines.filter(machinemodel_id__doors
= doors)

        body_style = self.request.GET.get('body_style')
        if body_style:
            machines =
machines.filter(machinemodel_id__body_style = body_style)
        transmission = self.request.GET.get('transmission')
        if transmission:
            machines = machines.filter(transmission =
transmission)

        conditioner = self.request.GET.get('conditioner')
        if conditioner:
            machines = machines.filter(conditioner =
conditioner)

        fuel_type = self.request.GET.get('fuel_type')
        if fuel_type:
            machines = machines.filter(fuel_type = fuel_type)
        city = self.request.GET.get('city')

```

```

        if city:
            city = City.objects.get(id = city)
            machines =
machines.filter(machinemodel_id__body_style = body_style)
            flexible = self.request.GET.get('flexible')
            if flexible:
                machines = machines.filter(flexible = 1)
            production = self.request.GET.get('production')
            if production:
                machines = machines.filter(production_year__gte =
production)

            score = self.request.GET.get('score')
            if score:
                machines = machines.filter(score__gte =
production)

            machines = reviews.order_by('-score')
            return machines

    def get_context_data(self, **kwargs):
        result_context = super(MachineSearchListView,
self).get_context_data(**kwargs)
        producer = self.request.GET.get('producer')
        result_context ['producer'] = producer
        result_context ['producers'] = Producer.objects.all()
        machinemodel = self.request.GET.get('machinemodel')
        result_context ['machinemodel'] = machinemodel
        result_context ['machinemodels'] =
Machinemodel.objects.all()
        places = self.request.GET.get('places')
        result_context ['places'] = places
        doors = self.request.GET.get('doors')
        result_context ['doors'] = doors
        body_style = self.request.GET.get('body_style')
        result_context ['body_style'] = body_style
        transmission = self.request.GET.get('transmission')
        result_context ['transmission'] = transmission
        conditioner = self.request.GET.get('conditioner')
        result_context ['conditioner'] = conditioner
        fuel_type = self.request.GET.get('fuel_type')

```

```
result_context ['fuel_type'] = fuel_type
city = self.request.GET.get('city')
result_context ['city'] = city
result_context ['cities'] = City.objects.all()
result_context ['countries'] = Country.objects.all()
flexible = self.request.GET.get('flexible')
result_context ['flexible'] = flexible
production = self.request.GET.get('production')
result_context ['production'] = production
score = self.request.GET.get('score')
result_context ['score'] = score
return result_context
```

**ДОДАТОК В**  
**Слайди презентації**

Національний університет «Запорізька політехніка»

кафедра програмних засобів

Дипломна кваліфікаційна робота

**ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ НАДАННЯ  
ВЛАСНИХ АВТОМОБІЛІВ У ПРОКАТ**

Виконав

ст. гр. КНТ-128

І. Д. Шевченко

Керівник

к.т.н., доцент

А. Є. Казурова

2022

Рисунок В.1 – Слайд № 1

**Об'єкт роботи** – процес прокату автомобілів.

**Предмет роботи** – програмне забезпечення надання власних автомобілів у прокат.

**Мета роботи** – розробити програмне забезпечення надання власних автомобілів у прокат для підтримки взаємодії між власниками автомобілів і орендарями.

{ 2 }

Рисунок В.2 – Слайд № 2

## Завдання роботи

- аналіз предметної області;
- проєктування бази даних;
- проєктування програмного забезпечення надання власних автомобілів у прокат;
- програмна реалізація програмного забезпечення надання власних автомобілів у прокат.

{ 3 }

Рисунок В.3 – Слайд № 3

## Порівняльна характеристика програмних аналогів з виконуваною розробкою

| Критерій                                    | Turo         | Rentben  | OwnRentaler                  |
|---------------------------------------------|--------------|----------|------------------------------|
| Країна дії                                  | Фактично США | Угорщина | Україна, може бути розширено |
| Розширена кількість фільтрів пошуку         | Так          | Так      | Так                          |
| Реєстрація власників автомобілів            | Так          | Так      | Так                          |
| Відгуки                                     | Так          | Так      | Так                          |
| Визначення водіїв для керування автомобілем | Ні           | Ні       | Так                          |
| Визначення дефектів автомобілів             | Ні           | Ні       | Так                          |

{ 4 }

Рисунок В.4 – Слайд № 4

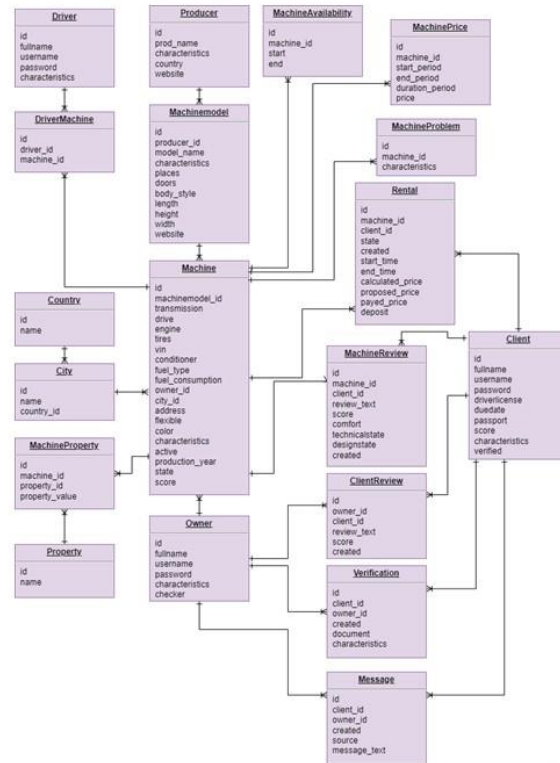
$$\begin{pmatrix} 5 \end{pmatrix}$$

## Діаграми прецедентів



Рисунок В.6 – Слайд № 6

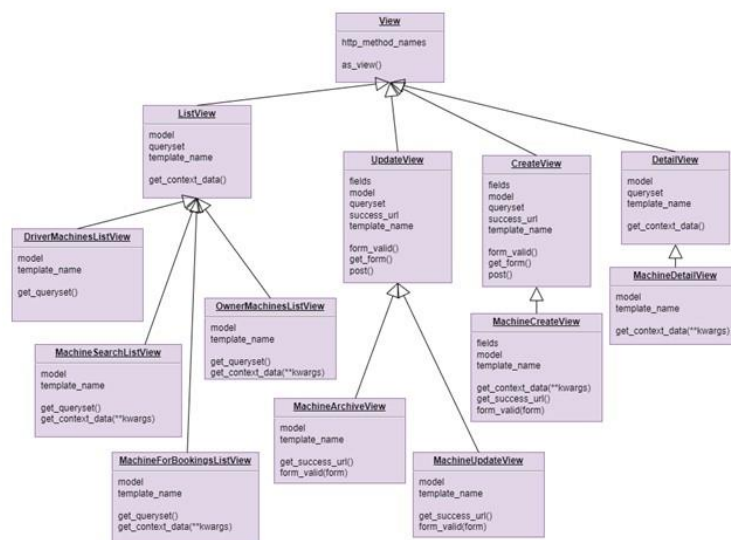
## Схема бази даних



7

Рисунок В.7 – Слайд № 7

## Діаграма класів



8

Рисунок В.8 – Слайд № 8

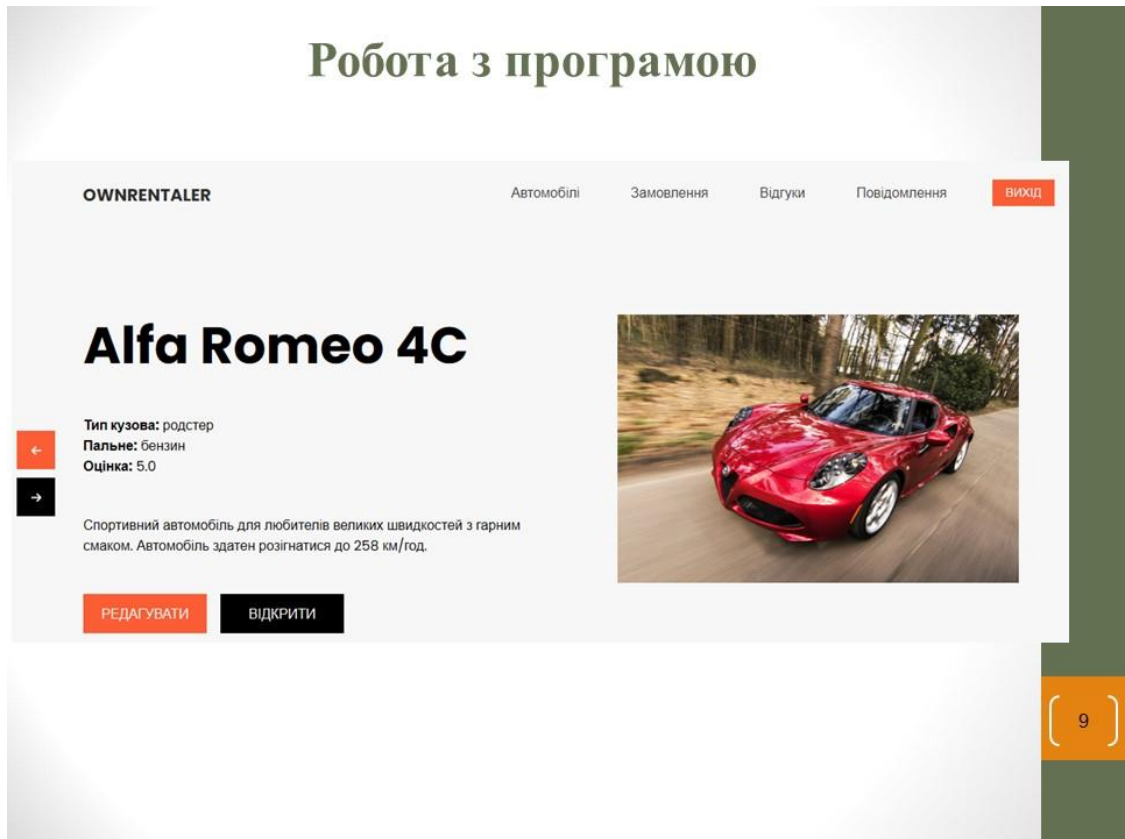


Рисунок В.9 – Слайд № 9

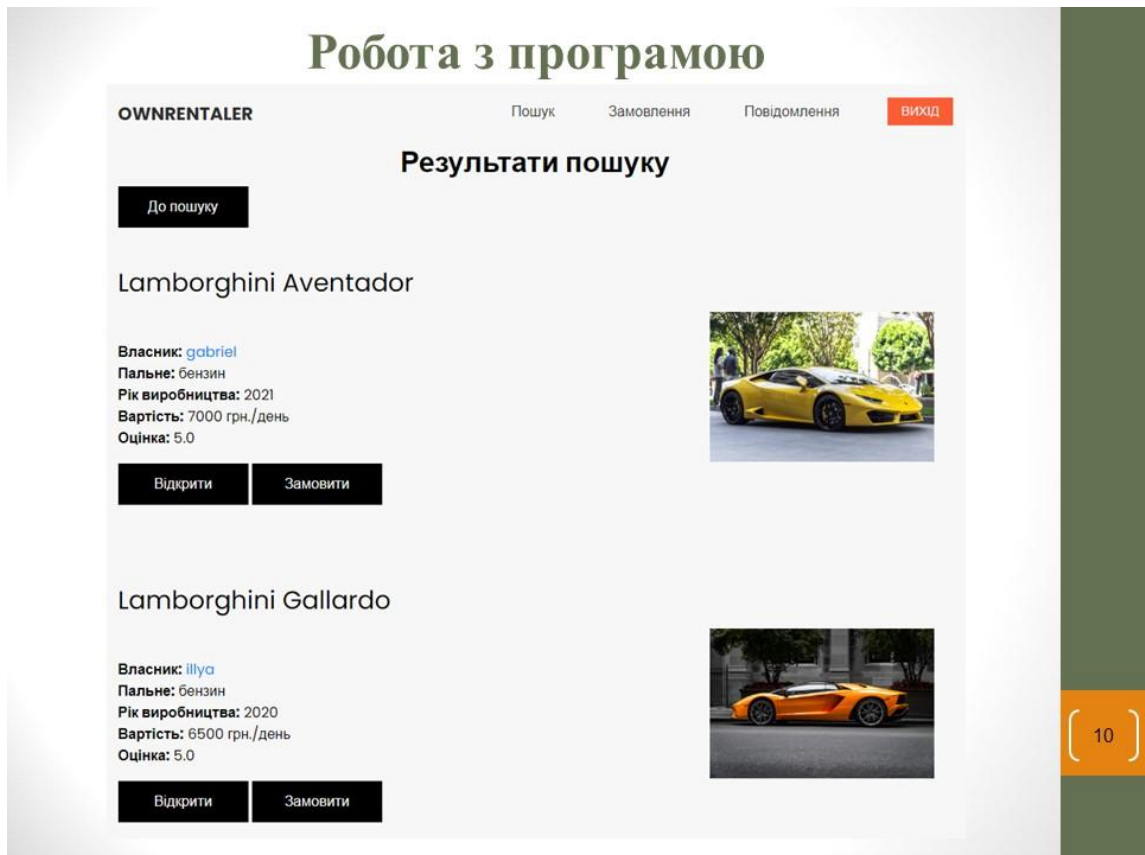


Рисунок В.10 – Слайд № 10

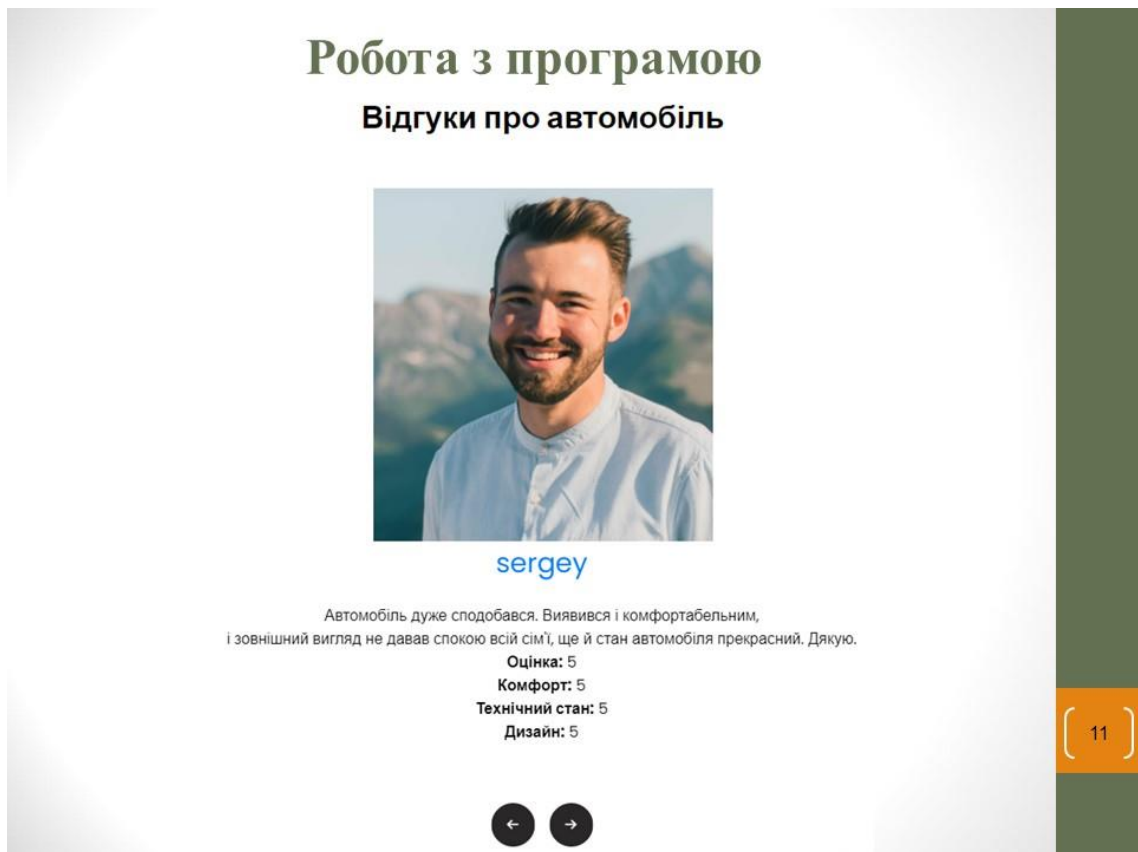


Рисунок В.11 – Слайд № 11

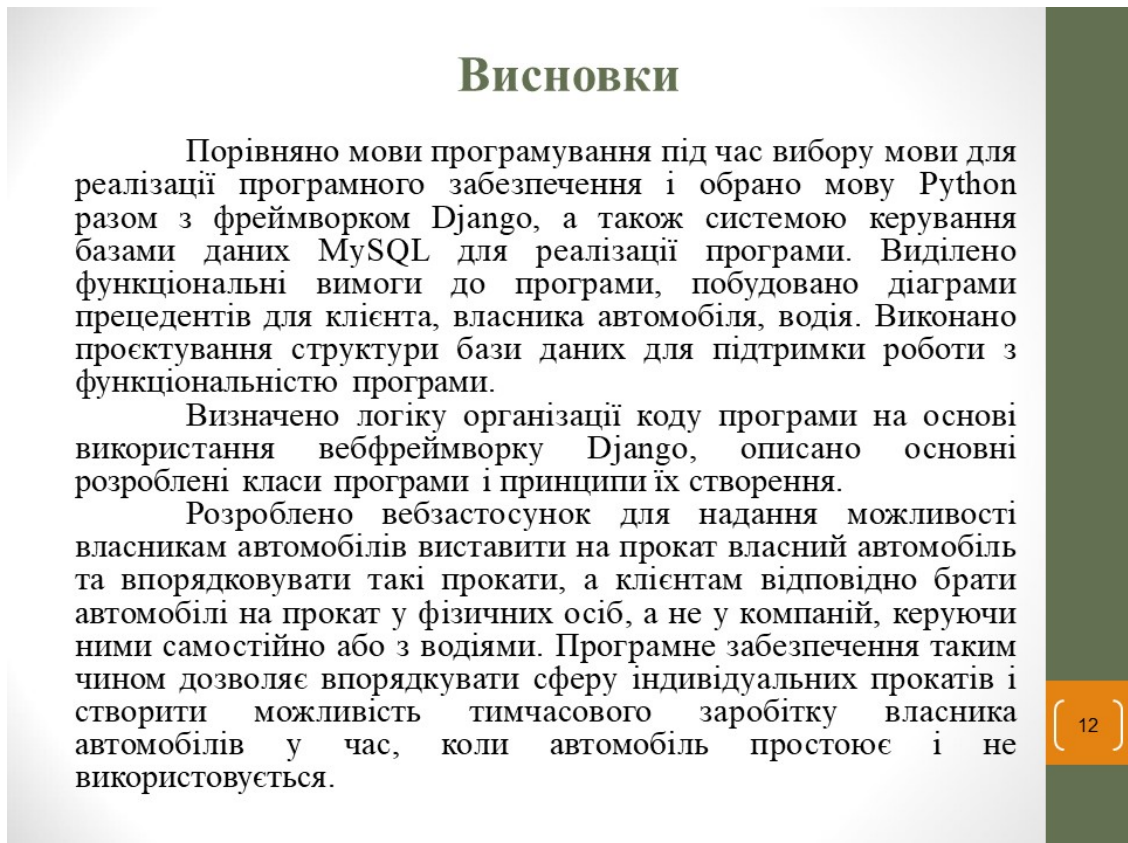


Рисунок В.12 – Слайд № 12