МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ВЕБОРІЄНТОВАНА ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ ОРЕНДИ АВТОМОБІЛІВ»

Рівень «бакалавр»
Галузь знань 12 «Інформаційні технології»,
спеціальність 122 «Комп’ютерні науки»
Виконав здобувач 4 курсу
Стоянов Артем Ігоревич

Керівник: к.пед.н., доцент
Лобода Юлія Геннадіївна

Рецензент: к.пед.н., доцент
Логінова Наталія Іванівна

Одеса – 2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет кібербезпеки та інформаційних технологій
Кафедра інформаційних технологій
Галузь знань: 12 Інформаційні технології
Спеціальність: 122 Комп'ютерні науки
Назва освітньої програми: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних технологій
доцент Н.І. Логінова
_____ «____ »_____ 20__ року

З А В Д А Н Н Я
на кваліфікаційну (бакалаврську) роботу
здобувачу Стоянову Артему Ігоревичу

1. Тема роботи: «Веборієнтована інформаційна система для оренди автомобілів»
Керівник роботи: к.пед.н, доцент Юлія Геннадіївна Лобода
затверджені наказом НУ «ОЮА» від «02» квітня 2024 року № 809-06

2. Строк подання здобувачем роботи «20» травня 2024 року

3. Дата видачі завдання «15» вересня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Формування функціональних вимог	13.10.2023	
2	Аналіз та вибір засобів розробки	25.10.2023	
3	Розробка архітектури системи	17.11.2023	
4	Проєктування бази даних	29.11.2023	
5	Розробка серверної частини	14.01.2024	
6	Розробка клієнтської частини засобами	15.03.2024	
7	Тестування функціональності сайту	17.04.2024	
8	Оформлення звітної частини	17.05.2024	

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота студента 4-го курсу Стоянова Артема Ігоревича на тему «Веборієнтована інформаційна система для оренди автомобілів» на здобуття першого (бакалаврського) рівня вищої освіти за спеціальністю 122 «Комп'ютерні науки», освітньою програмою «Комп'ютерні науки» містить 29 рисунків, 1 таблицю, 2 додатка, 20 літературних джерел за переліком посилань. Робота виконана на 77 сторінках загального тексту і 57 сторінках основного тексту.

Метою кваліфікаційної роботи є створення веборієнтованої інформаційної системи для оренди автомобілів, яка надає користувачам можливість швидко та зручно знаходити та орендувати автомобілі різних класів та комплектацій.

Об’єктом дослідження є веборієнтована інформаційна системи для оренди автомобілів.

Предмет дослідження є процес розробки веборієнтованої інформаційної системи для оренди автомобілів.

У даній роботі розроблено веборієнтовану інформаційну систему, яка дозволяє користувачам переглядати каталог автомобілів, реєструватися та авторизуватися на сайті, додавати автомобілі до списку улюблених, а також переглядати детальну інформацію про авто та умови їх оренди у модальному вікні. Досліджено специфіку розробки веборієнтованої інформаційної системи для сфери оренди, використовуючи інноваційні технології веб-розробки. Розроблену веборієнтовану інформаційну систему може бути використаний в різних сферах автомобільної індустрії, забезпечуючи зручну та доступну платформу для оренди автомобілів онлайн.

Ключові слова: веборієнтована інформаційна система, оренда автомобілів, інтерфейс, React, Frontend, Backend, Node.js, Express.js, MongoDB.

ABSTRACT

Qualification Work of Stoyanov Artem Ihorevych, 4th Year Student, on the Topic "WebOriented Information System for Car Rental" for Obtaining the First (Bachelor's) Level of Higher Education in the Specialty 122 "Computer Science", Educational Program "Computer Science"

The work contains 29 figures, 1 table, 2 appendices, and 20 literature sources listed in the references. The work comprises 77 pages of total text and 57 pages of main text.

The aim of this qualification work is to create a weboriented information system for car rental, which provides users with the ability to quickly and conveniently find and rent cars of various classes and configurations.

The object of the study is a weboriented information system for car rental.

The subject of the study is the process of developing a weboriented information system for car rental.

In this work, a weboriented information system has been developed, allowing users to view a car catalog, register and log in to the site, add cars to a favorites list, and view detailed information about cars and their rental conditions in a modal window. The specifics of developing a weboriented information system for the rental sector using innovative web development technologies were researched. The developed weboriented information system can be used in various areas of the automotive industry, providing a convenient and accessible platform for renting cars online.

Keywords: web-oriented information system, car rental, interface, React, Frontend, Backend, Node.js, Express.js, MongoDB.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

GUI - графічний інтерфейс користувача

JS - JavaScript

API - інтерфейс програмування застосунків

JWT - токен аутентифікації JSON

DOM - модель об'єктів документа

ВСТУП

Актуальність теми обумовлена постійним розвитком та розширенням сфери послуг автомобільної оренди в сучасному світі. З ростом мобільності населення зростає і попит на зручні та доступні сервіси оренди автомобілів. Це обумовлено не лише зростанням туристичного руху, але й розширенням ринку довгострокової оренди для бізнесу та приватних осіб.

Метою кваліфікаційної роботи є створення веборієнтованої інформаційної системи для оренди автомобілів, яка надає користувачам можливість швидко та зручно знаходити та орендувати автомобілі різних класів та комплектацій.

Об'єктом дослідження є веборієнтована інформаційна системи для оренди автомобілів.

Предмет дослідження є процес розробки веборієнтованої інформаційної системи для оренди автомобілів.

Мета досягнення поставленої роботи передбачає такі завдання:

– аналіз предметної області, виявлення та оцінка потреб клієнтів у сфері оренди автомобілів, а також визначення основних критеріїв для системи;

– формулювання вимог системи, таких як можливості реєстрації, авторизація користувача, можливість перегляду та оренди автомобілів;

– вибір технологій, а саме аналіз стеків технологій і вибір найкращих для реалізації завдання;

– розробка архітектури системи, створення інформаційної моделі для тематичної області, проектування баз даних і моделювання програмних систем з використанням відповідних підходів та інструментів;

– розробка серверних і клієнтських компонентів системи, інтерфейсу користувача та логіки функціоналу, а також тестування для перевірки якості та надійності продукту.

Для реалізації кваліфікаційної роботи використовуються методи наукового аналізу, спостереження та порівняння. Аналіз проводиться для визначення вимог до функціональності ситеми, для більш зручного використання. Спостереження та

порівняння використовуються для оцінки функціональних вимог системи та перевірки зручності її використання. Ці методи визначають ключові функції та можливості, які повинні бути в системі, щоб задовольнити потреби користувачів і підвищити її ефективність.

Практичне значення роботи є створення функціональної та ефективної інформаційної системи для оренди автомобілів, що відповідає сучасним вимогам та потребам користувачів. Розроблений додаток матиме практичне значення для автомобільних компаній, які надають послуги оренди, а також для користувачів, які шукають зручний та надійний спосіб оренди автомобіля.

Результати кваліфікаційного дослідження апробовані на IX Всеукраїнської науково-практичній конференції «Інформаційне суспільство: проблеми та перспективи» (м. Одеса, 24 травня 2024 р.) [1].

1 АНАЛІЗ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

1.1 Аналіз предметної області

Сучасні технології, які пов'язані з інформаційними процесами, відкривають нескінченні можливості для швидкого доступу до інформації, її отримання та передачі у будь-якому куточку світу, роботи з даними в інтернеті, організація процесу праці, складання звітів тощо.

Веборієнтована інформаційна система – це система, доступ до якої здійснюється за допомогою веббраузера через таку мережу, як Інтернет [2]. Вона автоматично збирає та оновлює свої дані за допомогою вказаних веб-сторінок, повідомляє про зміни використовуваних веб-сторінок, перетворює отримані дані в XML-документи, зберігає їх у XML-репозиторії, приймає запити та перетворює знайдені дані в інформацію [3].

Оренда автомобілів з кожним роком стає все популярнішою та активно розвивається. Багато людей віддають перевагу користуванню орендованим авто, аніж купівлі власного. Це пов'язано з перевагами, які надає така послуга: комфортні умови оренди, можливість оренди авто будь-якої марки, швидке бронювання та доставлення автомобілю. Веборієнтовані інформаційні системи служать помічниками користувачам, дозволяючи швидко вибрати потрібний автомобіль та легко оформити оренду.

Веборієнтована інформаційна система для оренди автомобілів – це інструмент, призначений для швидкої, зручної та ефективної оренди автомобілів.

Основними характеристиками веборієнтованої інформаційної системи оренди автомобілів є:

– використання найкращих інструментів та фреймворків, які дозволяють розробляти ефективні додатки;

– доступність до системи оренди автомобілів через веббраузери на різних пристроях;

– графічний інтерфейс користувача (GUI), який робить взаємодію з системою інтуїтивно зрозумілою та зручною;

– функціональність системи, яка надає широкий спектр функцій, таких як прокат, каталог автомобілів, реєстрація та авторизація користувачів та управління популярними оголошеннями;

– безпека, використання відповідних процедур шифрування та захисту даних для забезпечення конфіденційності даних, при аутентифікації користувачів та захист особистих даних.

У цій праці відповідальністю розробника є аналіз технічних вимог, створення найкращого можливого дизайну системи та використання сучасних технік та інструментів програмування.

Є вимога до простоти дизайну та акценту на зручності оренди автомобілів. Для покращення досвіду користувача з системою, основні функції повинні бути зручні та легко доступні.

Бізнеси також можуть заощадити гроші, використовуючи інформаційну систему для прокату автомобілів, оскільки потрібно менше фізичних точок продажу, а процес прокату оптимізується за допомогою широкого автоматизованого процесу.

Отже, для ефективного розвитку веборієнтованих інформаційних систем оренди автомобілів необхідний ретельний підхід, який включає аналіз потреб користувачів та бізнесу, використання сучасної технології та технік програмування, а також увагу до дизайну.

1.2 Аналіз наявних аналогів

У сучасному світі автомобільна прокатна індустрія набула значної популярності. До сфери її впливу потрапили не лише традиційні компанії, а й стартапи, що пропонують новаторські підходи до організації автомобільної оренди. У цьому розділі ми проаналізуємо три провідні програмні продукти у даній галузі: rentcars.com, car-rent.ua та rental.ua.

Rentcars.com

Rentcars.com є одним із найбільших сервісів з онлайн-прокату автомобілів у світі [4]. Цей сервіс пропонує широкий вибір автомобілів в різних країнах і містах, забезпечуючи зручність і доступні ціни для клієнтів.

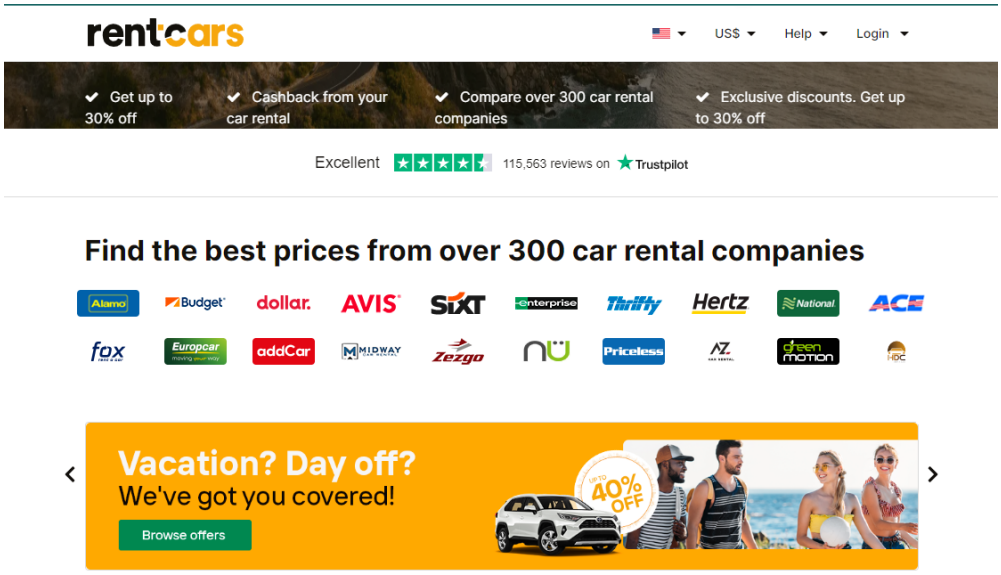

Рисунок 1.1 – Зображення інтерфейсу сайту rentcars.com

Основні переваги rentcars.com включають:

– Глобальне покриття: Rentcars.com працює у багатьох країнах світу, що дозволяє користувачам знаходити автомобілі для оренди практично в будь-якій точці планети;

– Зручний інтерфейс: Вебсайт та мобільний додаток rentcars.com мають інтуїтивно зрозумілий і легкий у використанні інтерфейс, що сприяє зручності користувачів під час бронювання автомобілів.

Недоліки:

– Значна конкуренція: У зв'язку з великим числом аналогічних сервісів, rentcars.com змушений боротися за свою аудиторію, що призводить до зниження маржі компанії;

– Складний процес реєстрації та авторизації.

Car-rent.ua

Car-rent.ua є одним з лідерів ринку автомобільної оренди в Україні [5]. Ця платформа спеціалізується на наданні послуг прокату автомобілів як для короткострокового, так і для довгострокового користування.

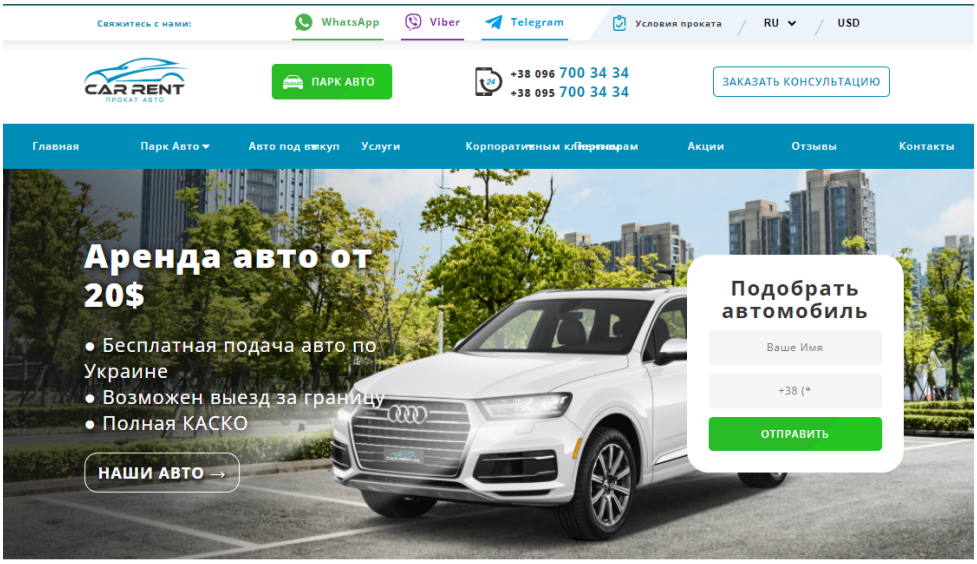

Рисунок 1.2 – Зображення інтерфейсу сайту car-rent.ua

Основні переваги car-rent.ua включають:

– Регіональна спеціалізація: Car-rent.ua сконцентрований на українському ринку, що дозволяє платформі краще розуміти потреби місцевих клієнтів та пропонувати їм більш індивідуалізовані послуги;

– Широкий вибір автомобілів: Платформа пропонує різноманітні типи автомобілів, що задовольняють потреби різних категорій клієнтів.

Недоліки:

– Обмежене покриття: У порівнянні зі своїми міжнародними конкурентами, car-rent.ua має обмежене географічне покриття, що обмежує можливості росту компанії.

Rental.ua

Rental.ua є одним зі стартапів, які активно розвиваються на ринку автомобільної оренди [6]. Цей сервіс відрізняється інноваційними підходами до

організації автомобільної прокату та акцентує увагу на високій якості обслуговування та персоналізованому підході до кожного клієнта.

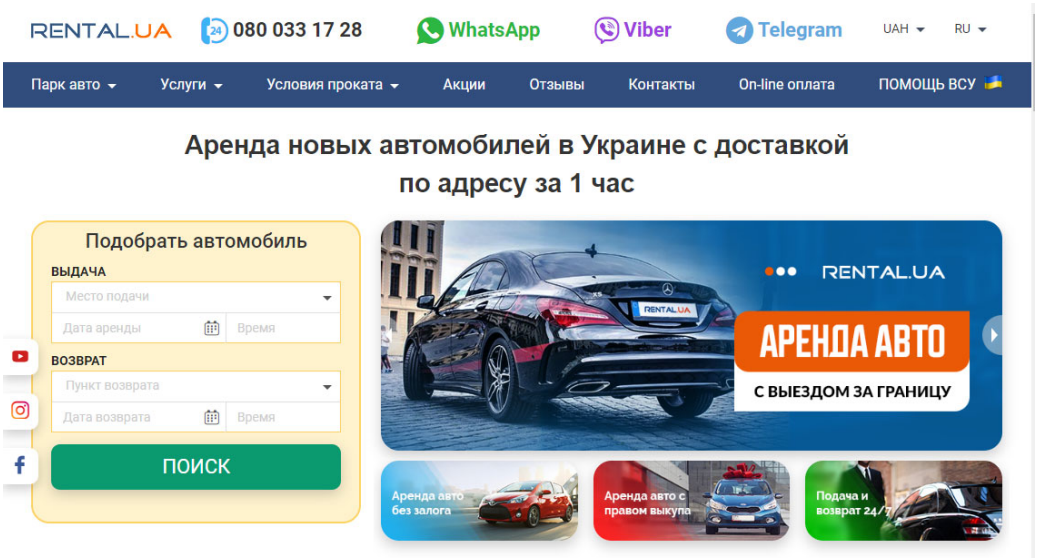

Рисунок 1.3 – Зображення інтерфейсу сайту rental.ua

Основні переваги rental.ua включають:

– Інноваційні рішення: Rental.ua використовує сучасні технології та аналіз даних для покращення процесів оренди автомобілів і надання клієнтам найкращого сервісу;

– Персоналізація: Сервіс надає індивідуальний підхід до кожного клієнта, враховуючи їхні потреби та побажання.

– Недоліки:

– Обмежене покриття: Обмежений вибір автомобілів у менших містах або регіонах;

– Інтерфейс менш інтуїтивний порівняно з іншими платформами.

Проаналізувавши програмні продукти (rentcars.com, car-rent.ua та rental.ua), робиться висновок: Rentcars.com пропонує широкий вибір автомобілів у багатьох країнах та зручний інтерфейс, але має значну конкуренцію та складний процес реєстрації. Car-rent.ua спеціалізується на українському ринку, пропонує широкий вибір автомобілів, але має обмежене географічне покриття. Rental.ua відрізняється

інноваційними підходами, персоналізованим сервісом, але має обмежений вибір автомобілів та інтерфейс менш інтуїтивний.

Таблиця 1.1 – SWOT-аналіз конкурентів

Аналіз	Rentcars.com	Car-rent.ua	Rental.ua
Сильні сторони	Глобальне покриття, зручний інтерфейс	Регіональна спеціалізація, широкий вибір автомобілів	Інноваційні рішення, персоналізація
Слабкі сторони	Значна конкуренція, складний процес реєстрації та авторизації	Обмежене географічне покриття	Обмежене покриття, менш інтуїтивний інтерфейс
Можливості	Розширення ринків, партнерські угоди	Розширення мережі прокату, вдосконалення сервісу	Розвиток технологій, партнерські відносини
Загрози	Зріст кількості конкурентів, зміни у правовому середовищі	Поява нових конкурентів, економічні труднощі	Фінансові обмеження, конкуренція з боку вже відомих брендів

Дана веб-орієнтована інформаційна система для оренди автомобілів має наступні переваги**:**

- інтерфейс зрозумілий та легкий у використанні;
- включає затребувані функції, такі як можливість фільтрувати оголошення за різними критеріями, додавати оголошення в улюблені та порівнювати різні автомобілі;
- зберігає улюблені авто, унікальні для кожного користувача;
- включає простий та зручний процес реєстрації та авторизації.

1.3 Визначення функціональних вимог та виявлення варіантів використання веборієнтованої інформаційної системи

1.3.1 Визначення функціональних вимог

Під час розробки веборієнтованої інформаційної системи для оренди автомобілів визначено функціональні вимоги, які задовольняють потреби користувачів та забезпечують зручність та ефективність використання системи.

Система розроблена з адаптивним дизайном для можливості максимально забезпечити його працездатність на всіх типах пристроїв та екранних розмірах. Особливості функціональних вимог:

1) Домашня сторінка:
   a) Користувачам надається опис пропонованих компанією послуг;
   b) Забезпечено перехід до інших сторінок системи завдяки навігаційному меню.
2) Реєстрація:
   a) Форма реєстрації має поля для імені, електронної пошти та паролю;
   b) Валідація даних перевіряє правильність введеної інформації перед її відправкою на сервер;
   c) Створюється новий профіль користувача.
3) Авторизація:
   a) Форма авторизації має поля для введення електронної пошти та паролю;
   b) Валідація даних перевіряє правильність введеної інформації перед її відправкою на сервер;
   c) Зареєстровані користувачі можуть користуватися повним набором функціоналу системи.
4) Каталог автомобілів:
   a) Відображається список доступних для оренди автомобілів з базовою інформацією про них;
   b) Натискання на кнопку "Load more" дозволяє завантажити більше оголошень;
   c) Є можливість фільтрувати оголошення за маркою, ціною та пробігом, завдяки фільтру.
5) Улюблені оголошення:
   a) Зареєстрований користувач має можливість зберігати обрані ним оголошення;

b) Зареєстрований користувач має можливість додавати та видаляти оголошення зі списку улюблених;

c) Система зберігає стан списку улюблених оголошень при оновленні сторінки;

d) Вбудована унікальність гарантує, що список улюблених оголошень є унікальним для кожного користувача.

6) Модальне вікно з деталями:

a) Модальне вікно відображає детальну інформацію про обраний автомобіль та умови його оренди.

b) Натиснувши на "хрестик", клавішу Esc або backdrop, закривається модальне вікно.

7) Навігація:

a) Навігаційне меню забезпечує зручний перехід між різними сторінками системи.

b) У разі введення неправильного маршруту користувач автоматично потрапляє на домашню сторінку.

1.3.2 Варіанти використання програмного продукту

Розроблена веборієнтована система має задовольняти різноманітним потребам користувачів, які можна умовно розділити на наступні варіанти використання:

1. Реєстрація нового користувача:

a) Користувач відвідує сторінку реєстрації та заповнює необхідні поля;

b) Після успішної реєстрації користувач отримує доступ до особистого кабінету.

2. Авторизація користувача:

a) Користувач вводить свої дані (електронну пошту та пароль) на сторінці авторизації;

b) Після успішної авторизації користувач отримує доступ до особистого кабінету.

3. Перегляд каталогу автомобілів:

a) Користувач переходить на сторінку каталогу, де відображається перелік доступних автомобілів;

b) Користувач відкриває докладну інформацію про кожен автомобіль, використовуючи кнопку "Learn more";

c) Користувач вводить у спеціальні інпути марку авто, ціну або пробіг та система фільтрує авто за цими показниками.

4. Додавання до улюблених:

a) Користувач додає оголошення до свого списку улюблених, клікнувши на відповідну кнопку на картці автомобіля;

b) Обрані оголошення будуть доступні на сторінці "Улюблені";

c) Користувачу будуть доступні лише його унікальні обрані оголошення.

5. Перегляд улюблених оголошень:

a) Користувач переглядає свої обрані оголошення на відповідній сторінці;

b) Можливість видалення оголошення зі списку улюблених.

6. Модальне вікно з деталями автомобіля:

a) Користувач отримує детальну інформацію про автомобіль, натиснувши на кнопку "Learn more" на картці автомобіля.

7. Навігація по системі:

a) Користувач легко переміщається між різними сторінками системи за допомогою навігаційного меню.

8. Зв'язок з компанією:

a) Користувач звертається до компанії за допомогою кнопки "Rental car" та надати номер телефону для зв'язку.

Розроблена веборієнтована система покликаний забезпечити зручний та функціональний інтерфейс для користувачів, що дозволить їм з легкістю користуватися послугами компанії з оренди автомобілів.

1.4 Вибір засобів розробки

1.4.1 Засоби розробки бази даних застосунку

В процесі розробки веб-орієнтованої інформаційної системи для оренди автомобілів, одним із ключових аспектів є вибір засобів для роботи з базою даних. З урахуванням особливостей проєкту, були обрані такі інструменти.

MongoDB — це база даних документів, розроблена для полегшення розробки та масштабування програм. Вона була обрана для керування базами даних [7].

Основні можливості MongoDB:

- Сховище, орієнтоване на JSON-подібну схему даних;
- Досить гнучка мова для формування запитів;
- Повна підтримка індексів для швидкого доступу до даних;
- Профілювання запитів для кращої продуктивності;
- Ефективне зберігання таких даних, як фото та відео;
- Ведення журналу операцій для моделювання баз даних.

Для зручного взаємодії з MongoDB з рівня Node.js була обрана бібліотека Mongoose. Вона дозволяє розробникам визначати свої моделі даних, використовуючи підхід на основі схеми, і надає багатий набір функцій, які спрощують процес роботи [8].

Використоавується Mongoose для роботи з MongoDB, для конструктору запитів, перевірки даних та визначення структури цих даних.

Node.js — це безкоштовне міжплатформне середовище виконання JavaScript із відкритим кодом, яке дозволяє розробникам створювати сервери, веб-програми, інструменти командного рядка та сценарії. Node.js має унікальну перевагу, оскільки мільйони інтерфейсних розробників, які пишуть JavaScript для браузера,

тепер можуть писати код на стороні сервера на додаток до коду на стороні клієнта без необхідності вивчати абсолютно іншу мову[9].

Frontend розробляється на JavaScript за допомогою React, використання Node.js дозволить максимально ефективно використовувати загальний досвід команди.

Express.js - це мінімалістичний та гнучкий фреймворк для веб-застосунків, побудованих на Node.js, що надає широкий набір функціональності. Маючи в свому розпорядженні безліч допоміжних HTTP-методів та проміжних обробників, створювати надійні API можна легко і швидко[10].

Використання Express.js дозволяє швидко розробляти потужний та надійний API для системи для оренди автомобілів.

JsonWebToken (JWT) — це компактний, безпечний для URL-адрес, відкритий стандарт, який визначає метод безпечної передачі інформації шляхом кодування та підпису даних JSON. JWT використовуються для передачі різних типів інформації, але в основному його використовують для доставки «вимог», які є фрагментами інформації. На практиці вимоги, швидше за все, міститимуть корисну інформацію про користувача, що найчастіше використовується для авторизації та автентифікації[11].

Використання JWT дозволить ефективно керувати сеансами користувачів та забезпечити безпеку нашого API.

bcrypt — адаптивна криптографічна функція формування ключа, що використовується для безпечного зберігання паролів. Для захисту від атак за допомогою райдужних таблиць bcrypt використовує сіль (salt); крім того, функція є адаптивною, час її роботи легко налаштовується і її можна сповільнити, щоб ускладнити атаки перебором [12].

Використання bcrypt дозволить забезпечити безпеку паролів користувачів у базі даних, забезпечуючи їхню захищеність від зловмисників.

cors - це middleware для Express.js, яке дозволяє обробляти запити CORS (Cross-Origin Resource Sharing). CORS — це механізм безпеки, який дозволяє

серверу вказати, які джерела мають доступ до ресурсів в веб браузері та їх завантаження[13].

Використання cors дозволить забезпечити доступ до мого API з веб-сторінок, розташованих на різних доменах, забезпечуючи при цьому безпеку від зловмисників.

express-validator - це middleware для Express.js, яке дозволяє здійснювати перевірку та валідацію даних, отриманих від клієнта. Використання express-validator дозволить перевіряти правильність даних, отриманих від користувачів, перед їхнім збереженням у базі даних, що забезпечить цілісність та надійність системи.

Обираючи ці засоби розробки для backend частини нашого додатку для оренди автомобілів, ми покладаємося на їхню надійність, ефективність та зручність використання. Вони дозволять нам швидко розробити потужний та безпечний API, який задовольнить потреби наших користувачів і забезпечить їм комфортне використання нашого додатку.

1.4.3 Засоби розробки frontend частини

Під час розробки frontend частини інформаційної системи для оренди автомобілів було вирішено використовувати JavaScript-фреймворк React. React.js – це бібліотека JavaScript для відтворення інтерфейсів користувача. Інтерфейс користувача складається з невеликих елементів, таких як кнопки, текст і зображення. React дозволяє вам об’єднувати їх у багаторазово використовувані вкладені компоненти. Від веб-сайтів до програм для телефону, все на екрані можна розбити на компоненти. [14]. Також він має такі переваги[15]:

1) Використання віртуального DOM та ефективного алгоритму оновлення дозволяє робити мінімальні зміни в реальному DOM, що покращує продуктивність застосунків;

2) JS React базується на компонентній архітектурі, що дозволяє розбивати інтерфейс на незалежні компоненти. Це спрощує розробку, тестування та

підтримку коду, оскільки компоненти можуть бути повторно використані та легко змінюються;

3) React пропагує односторонній потік даних, що сприяє простоті та передбачуваності управління станом додатків, полегшує відлагодження та тестування застосунків;

4) спільнота розробників, що використовує React велика та активна. Є безліч ресурсів, бібліотек та інструментів для розробки. Також існує багато сторонніх бібліотек, які підтримують React і допомагають розширити його функціональність.

Для керування станом застосунку вирішено використовувати бібліотеку Redux.

Redux – це JavaScript-бібліотека, покликана спростити управління станом вашого веб-додатку. Її основне призначення полягає в тому, щоб зробити управління даними більш організованим і передбачуваним.

У центрі концепції Redux знаходиться сховище стану (Store). Це своєрідне сховище, де зібрані всі дані вашого застосунку. Він як спільний банк даних, до якого можна звернутися з будь-якої точки застосунку.

Redux дає змогу ефективно управляти даними, роблячи процес розроблення більш організованим і передбачуваним.

Принцип того, як працює Redux, ґрунтується на таких концепціях:

– Application state: це об’єкт, який містить усі дані, необхідні для роботи програми;

– Actions: це об’єкти, які представляють зміни стану додатка;

– Reducers: це функції, які визначають, як стан програми змінюється у відповідь на дії [16].

Для управління маршрутами та перехідами між сторінками інформаційної системи використовується бібліотека React Router. React Router — це API для веб-додатків, які використовують бібліотеку React. Маршрутизатор React використовує динамічну маршрутизацію (тобто маршрутизацію, яка здійснюється під час рендерингу вашої програми, а не в конфігурації додатка).

Маршрутизатор React та динамічна маршрутизація на стороні клієнта дозволяє створити односторінковий веб-додаток з навігацією, не оновлюючи сторінку, під час навігації користувача. Для виклику компонентів маршрутизатор React використовує їх структуру, а далі компоненти відображають відповідну інформацію [17].

Усі ці інструменти разом дозволять створити потужну та динамічну frontend частину інформаційної системи для оренди автомобілів, яка буде забезпечувати зручну та ефективну взаємодію з користувачами.

1.5 Маркетингові інструменти в створенні та просуванні програмного продукту

Створення і розвиток програмного продукту, особливо у сфері інформаційних технологій, потребує не лише технічних навичок, а й уміння ефективно використовувати маркетингові інструменти для залучення користувачів та просування продукту на ринку. У цьому розділі розглядається різноманітні аспекти маркетингу, включаючи аналіз ринку, визначення цільової аудиторії, пошук конкурентних переваг, а також використання інструментів SEO (Search Engine Optimization) та SMM (Social Media Marketing) для успішного просування веборієнтованої інформаційної системи для оренди автомобілів;

– Аналіз ринку: Першим кроком у створенні ефективної маркетингової стратегії є проведення докладного аналізу ринку. Для мого продукту, системи для оренди автомобілів, маємо ретельно дослідити поточну ситуацію на ринку оренди автомобілів, включаючи основних гравців, їх послуги та цінову політику. Також важливо визначити тенденції ринку, популярність певних типів автомобілів серед клієнтів, а також особливості споживчого попиту;

– Визначення цільової аудиторії: Після проведення аналізу ринку можно приступити до визначення цільової аудиторії нашого продукту. Оскільки система орієнтована на оренду автомобілів, можно визначити своєю цільовою аудиторією людей, які шукають тимчасове транспортне засоби для подорожей, відпочинку або

бізнесу. Така аудиторія включає як індивідуальних клієнтів, так і компанії, які потребують автомобілів для службових поїздок;

– Пошук конкурентних переваг: Одним із ключових аспектів успішного просування на ринку є визначення конкурентних переваг продукту. Система для оренди автомобілів виділяється за рахунок таких факторів, як широкий вибір автомобілів різних марок і моделей, зручний інтерфейс користувача, доступні ціни та легка система бронювання та оплати;

– Використання інструментів SEO: Пошукова оптимізація сайту або ж SEO — процес коригування HTML-коду, текстового наповнення (контенту), структури сайту, контроль зовнішніх чинників для відповідності вимогам алгоритму пошукових систем, з метою підняття позиції сайту в результатах пошуку в цих системах за певними запитами користувачів. Чим вище позиція сайту в результатах пошуку, тим більша ймовірність, що відвідувач перейде на нього з пошукових систем, оскільки люди зазвичай йдуть за першими посиланнями[18]. SEO є важливим інструментом для забезпечення видимості нашого сайту в пошукових системах, таких як Google. Для досягнення успіху в цьому напрямку можно використовувати ключові слова та фрази, пов'язані з орендою автомобілів, в оптимізації вмісту нашого сайту, а також в інших маркетингових матеріалах;

– Використання інструментів SMM: Маркетинг у соціальних мережах, або — комплекс заходів щодо використання соціальних медіа як каналів для просування компаній та вирішення інших бізнес-завдань [19]. SMM дозволяє залучати нових користувачів та підтримувати зв'язок зі своєю аудиторією через соціальні медіа платформи. Можно публікувати цікавий та корисний контент про послуги, проводити акції та розіграші, а також спілкуватися з клієнтами через коментарі та приватні повідомлення.

Маркетинг є невід'ємною частиною процесу створення та просування програмного продукту. Використання різноманітних маркетингових інструментів, таких як аналіз ринку, визначення цільової аудиторії, пошук конкурентних переваг, SEO та SMM, дозволяє досягти успіху на конкурентному ринку програмного забезпечення. У нашому випадку, створення і просування веборієнтованої

інформаційної системи для оренди автомобілів, ці інструменти грають важливу роль у залученні користувачів та популяризації нашого продукту.

1.6 Висновки до розділу 1

Під час аналізу предметної області та вибору засобів розробки було проведено глибоке дослідження сучасних тенденцій у сфері оренди автомобілів та розробки веб-додатків. Виявлено, що існуючі аналоги мають певні обмеження та недоліки, які потребують вирішення. SWOT-аналіз конкурентів дозволив з'ясувати, що є можливості для створення конкурентоздатного продукту, який задовольнить потреби користувачів більш ефективно.

Розроблені функціональні вимоги та варіанти використання показали, які основні функції повинен виконувати веборієнтованій системі, та вказали на основні сценарії взаємодії користувачів з ним.

Під час вибору засобів розробки було узято до уваги не лише їх функціональні можливості, а й простоту використання, підтримку спільнотою, швидкодію та інші технічні характеристики.

Маркетингові інструменти дозволили зрозуміти, які кроки слід буде вжити для просування продукту на ринку та залучення цільової аудиторії.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1 Інформаційна модель предметної області

Як було зазначено, веборієнтована інформаційна система для оренди автомобілів - це програмний комплекс, що дозволяє користувачам шукати, бронювати та орендувати автомобілі.

ER-модель є одним зі зручних інструментів уніфікованого подання даних.

Концептуальні ER-моделі не враховують особливості конкретної системи керування базами даних (СКБД). На базі концептуальної моделі можна побудувати фізичну діаграму, в якій вже будуть враховуватися особливості СКБД [20, с.7]. Модель для веборієнтованої інформаційної системи зображено ма рисунку 2.1.

Система складається з двох основних компонентів:

Frontend: Цей компонент відповідає за взаємодію користувачів з системою. Він написаний на JavaScript, React та Redux і включає в себе такі компоненти, як:

– Домашня сторінка: містить загальний опис послуг, що надаються компанією;

– Сторінка реєстрації: дозволяє користувачам створювати нові облікові записи;

– Сторінка авторизації: дозволяє користувачам входити до своїх облікових записів;

– Каталог автомобілів: відображає список доступних для оренди автомобілів. Користувачі можуть фільтрувати автомобілі за маркою, моделлю, ціною та іншими критеріями;

– Сторінка улюблених: відображає список автомобілів, які користувач додав до своїх улюблених;

– Модальне вікно: з'являється при натисканні на кнопку "Дізнатися більше" на картці оголошення. Воно містить детальну інформацію про автомобіль та умови його оренди.

Backend: Цей компонент відповідає за зберігання даних та бізнес-логіку системи. Він написаний на Node.js та використовує базу даних MongoDB. Backend включає в себе такі компоненти:

– Маршрутизатори: відповідають за обробку HTTP-запитів і надсилання відповідей;
– Контролери: реалізують бізнес-логіку системи;
– Моделі: представляють дані системи;
– З'єднання з базою даних: забезпечує доступ до бази даних MongoDB.

У системі оренди автомобілів є такі сутності:

Користувач: Користувач - це людина, яка реєструється авторизується, шукає, бронює та орендує автомобілі. Користувач має такі характеристики:

– id: Унікальний ідентифікатор користувача;
– fullName: Повне ім'я користувача;
– email: Адреса електронної пошти користувача;
– passwordHash: Хеш пароля користувача;
– createdAt: Дата та час створення запису користувача;
– updatedAt: Дата та час останнього оновлення запису користувача.

Автомобіль: Автомобіль - це транспортний засіб, який можна орендувати. Автомобіль має такі характеристики:

– id: Унікальний ідентифікатор автомобіля;
– year: Рік випуску автомобіля;
– make: Марка автомобіля;
– model: Модель автомобіля;
– type: Тип автомобіля (наприклад, SUV, седан, хетчбек);
– img: URL-адреса зображення автомобіля;;
– description: Опис автомобіля
– fuelConsumption: Витрата палива автомобіля;
– engineSize: Розмір двигуна автомобіля;
– accessories: Список аксесуарів, які доступні в автомобілі;
– functionalities: Список функцій, які доступні в автомобілі;

– rentalPrice: Ціна оренди автомобіля за день;

– rentalCompany: Назва компанії, яка здає автомобіль в оренду;

– address: Адреса компанії, яка здає автомобіль в оренду;

– rentalConditions: Умови оренди автомобіля;

– mileage: Пробіг автомобіля.

Улюблені: Це сутність, яка представляє собою список автомобілів, які користувач додав до своїх улюблених. Кожна сутність "Улюблені" матиме такі атрибути:

– userId: Ідентифікатор користувача, який додав автомобіль до своїх улюблених;

– carId: Ідентифікатор автомобіля, який був доданий до улюблених.

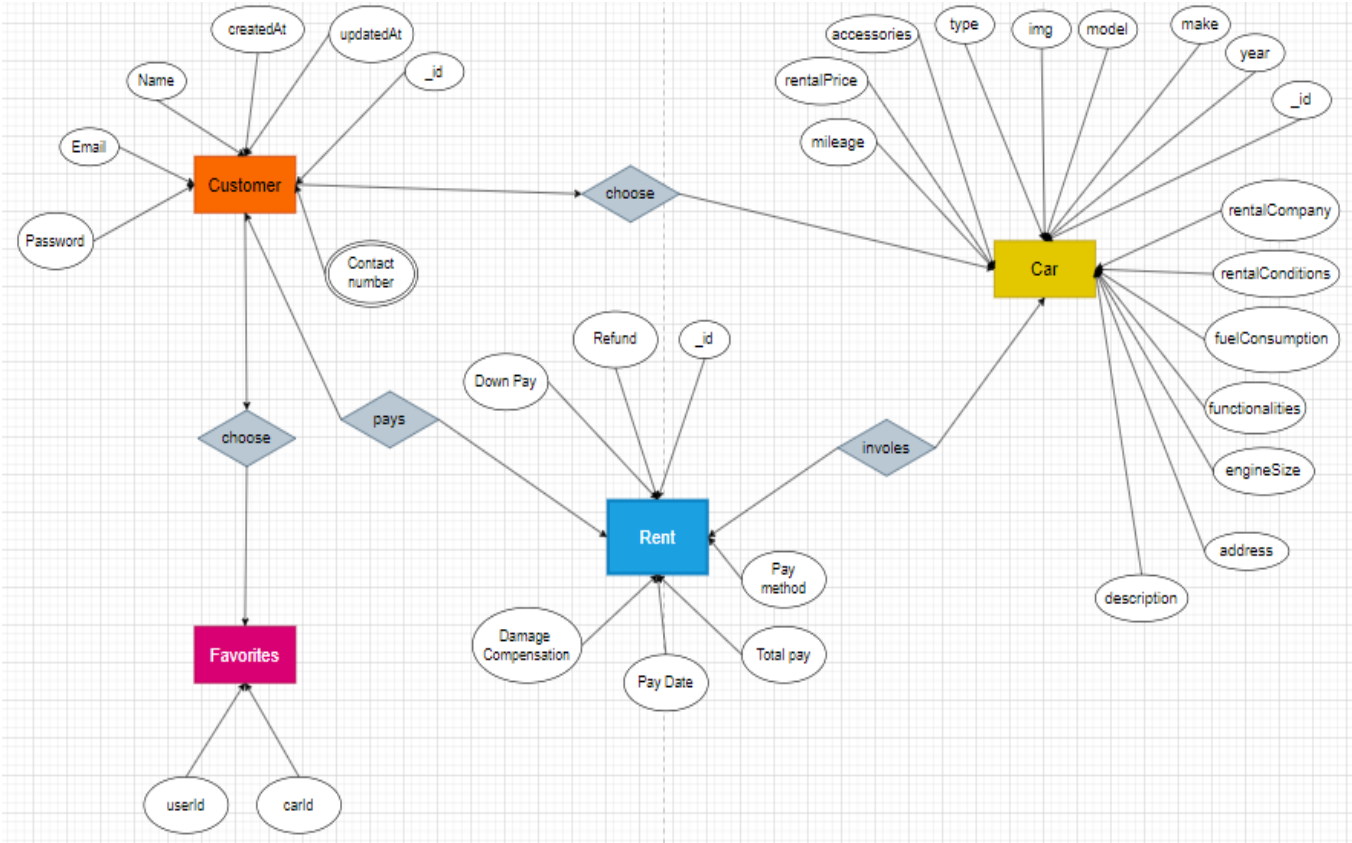

Рисунок 2.1 – ER – інформаційна модель

2.2 Архітектура програмного застосунку

Frontend:

Використовується React з Redux для створення вебінтерфейсу користувача. Основною ідею було розділення додатку на компоненти, які будуть відповідати за відображення окремих частин інтерфейсу.

Компоненти:

– CarList: Відображення списку автомобілів;

– CarSearch: Форма пошуку автомобілів за параметрами;

– Card: Карточка автомобіля з кнопками вподобання та детальною інформацією;

– Header: Верхній рядок із навігацією;

– InfiniteSlider: Слайдер для відображення банерів чи акцій;

– Loader: Візуальний ефект завантаження;

– Modal: Модальне вікно з детальною інформацією про автомобіль.

Redux Store:

– auth: Стан авторизації користувача;

– catalog: Список автомобілів та їх властивості;

– favorites: Список улюблених автомобілів;

– filter: Фільтри для каталогу автомобілів.

Сервіси:

– API: Модуль для взаємодії з сервером, відправка запитів і отримання даних;

– App: Головний компонент, який буде включати в себе всі інші компоненти та встановлювати маршрути за допомогою React Router.

Backend:

Використовується Node.js з Express для створення веб-сервера, який буде обробляти HTTP-запити від frontend та взаємодіяти з базою даних MongoDB.

Маршрутизатори:

– /auth/register: Маршрути для реєстрації користувачів;

– /auth/login: Маршрути для авторизації користувачів;

– /catalog: Маршрути для отримання та створення автомобілів.

Контролери:

– UserController: Обробка запитів, пов'язаних з користувачами;

– CarsCatalog: Обробка запитів, пов'язаних з каталогом автомобілів.

Моделі:

– User: Структура та методи для роботи з користувачами;

– Car: Структура та методи для роботи з автомобілями.

Валідація:

– registerValidation: Перевірка даних при реєстрації користувача;

– loginValidation: Перевірка даних при вході користувача;

– carsCatalogValidation: Перевірка даних при створенні нового автомобіля;

index.js: Головний файл сервера, в якому ми будемо налаштовувати Express, підключати маршрути та встановлювати зв'язок з базою даних.

Комунікація між frontend та backend:

Для забезпечення комунікації між frontend та backend були використані HTTP-запити. Ось основні запити:

– POST ('/auth/register') - запит на сервер з введеними даними для реєстрації користувачів;

– POST ('/auth/ login') - запит на сервер з введеними даними для авторизації користувачів;

– POST(' /catalog’) - запит на сервер з введеними даними для створення автомобілів;

– GET(' /catalog’) - запит на сервер з введеними даними для отримання автомобілів.

Для зручності ми будемо використовувати бібліотеку Axios на frontend для здійснення HTTP-запитів до нашого сервера.

Збереження даних:

Для збереження даних була використовована база даних MongoDB для забезпечення гнучкої та високої швидкості роботи з колекцією автомобілів.

Для взаємодії з базою даних була використована бібліотека Mongoose, яка дозволяє створювати моделі даних та виконувати запити до бази даних з frontend.

2.3 Проектування бази даних

Діаграма потоків даних (англ. Data Flow Diagram) – це діаграми, на яких відображаються потоки даних, процеси перетворення вхідних потоків на вихідні, сховища інформації, джерела і споживачі інформації, зовнішні щодо системи. [20, с.77].

Зовнішні сутності:

– користувач: Реєструється, авторизується, переглядає каталог авто, додає оголошення в улюблені, оновлює сторінку;

– адміністратор: Додає нові авто в каталог, редагує існуючі, видаляє авто.

Процеси:

Реєстрація користувача:

– Вхідні дані: Ім'я, email, пароль;

– Вихідні дані: Повідомлення про успішну/невдалу реєстрацію, токен користувача.

Авторизація користувача:

– Вхідні дані: Email, пароль;

– Вихідні дані: Повідомлення про успішну/невдалу авторизацію, токен користувача.

Перегляд каталогу авто:

– Вхідні дані: Марка авто, ціна, пробіг (необов'язково);

– Вихідні дані: Список авто, що відповідають заданим критеріям.

Додавання авто в улюблені:

– Вхідні дані: ID авто;

– Вихідні дані: Повідомлення про успішне/невдале додавання.

Додавання нового авто:

– Вхідні дані: Рік випуску, марка, модель, тип, URL зображення, опис, витрата палива, об'єм двигуна, аксесуари, функціональні можливості, ціна оренди, компанія орендодавця, адреса, умови оренди, пробіг;

– Вихідні дані: Повідомлення про успішне/невдале додавання.

Редагування авто:

– Вхідні дані: ID авто, нові дані про авто;

– Вихідні дані: Повідомлення про успішне/невдале редагування.

Видалення авто:

– Вхідні дані: ID авто;

– Вихідні дані: Повідомлення про успішне/невдале видалення.

Сховища даних:

– Користувачі: Зберігає інформацію про зареєстрованих користувачів (ID, ім'я, email, пароль, хеш пароля, дата створення, дата оновлення);

– Авто: Зберігає інформацію про авто в каталозі (ID, рік випуску, марка, модель, тип, URL зображення, опис, витрата палива, об'єм двигуна, аксесуари, функціональні можливості, ціна оренди, компанія орендодавця, адреса, умови оренди, пробіг);

– Улюблені оголошення: Зберігає інформацію про авто, додані користувачем в улюблені (ID користувача, ID авто).

Потоки даних:

1) Користувач -> Реєстрація користувача: Користувач вводить дані для реєстрації;

2) Реєстрація користувача -> Користувач: Система надсилає повідомлення про успішну/невдалу реєстрацію, токен користувача;

3) Користувач -> Авторизація користувача: Користувач вводить дані для авторизації;

4) Авторизація користувача -> Користувач: Система надсилає повідомлення про успішну/невдалу авторизацію, токен користувача;

5) Користувач -> Перегляд каталогу авто: Користувач вводить критерії пошуку (марка, ціна, пробіг);

6) Перегляд каталогу авто -> Користувач: Система надсилає список авто, що відповідають заданим критеріям;

7) Користувач -> Додавання авто в улюблені: Користувач надсилає ID авто, яке хоче додати в улюблені;

8) Додавання авто в улюблені -> Користувач: Система надсилає повідомлення про успішне/невдале додавання;

9) Додавання нового авто -> Адміністратор: Система надсилає повідомлення про успішне/невдале додавання;

10) Адміністратор -> Редагування авто: Адміністратор вводить ID авто та його нові дані;

11) Редагування авто -> Адміністратор: Система надсилає повідомлення про успішне/невдале редагування;

12) Адміністратор -> Видалення авто: Адміністратор надсилає ID авто для видалення;

13) Видалення авто -> Адміністратор: Система надсилає повідомлення про успішне/невдале видалення.

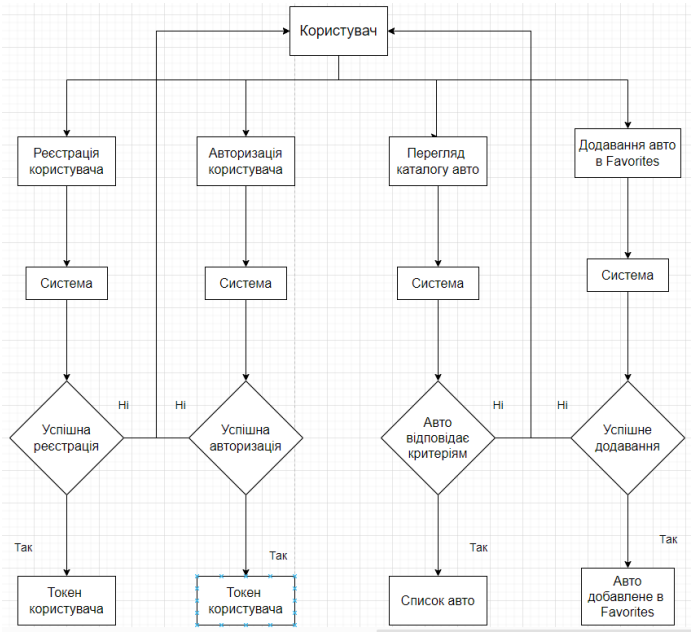

Рисунок 2.2 – діаграма DFD потоків даних для користувача

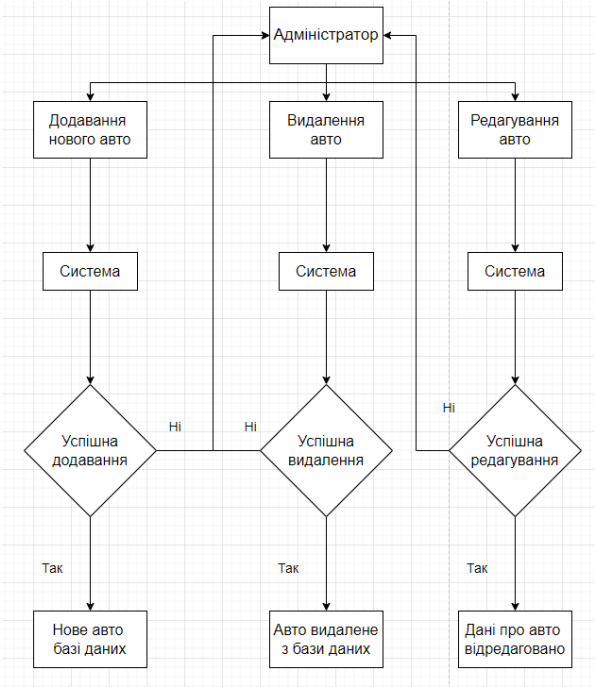

Рисунок 2.3 – діаграма DFD потоків даних для адміністратора

2.4 Моделювання програмної системи

Під моделювання програмної системи ми розуміємо процес створення веборієнтованої інформаційної системи за допомогою певних інструментів та методів.

Моделювання програмної системи дозволить краще розуміти процес створення ситеми, її функціонал та як користувач буде взаємодіяти з нею. Представлено у вигляді UML діаграм, таких як діаграма прецедентів, діаграма класів та діаграма діяльності.

Діаграма варіантів використання (або діаграма прецедентів, Use case diagram) є найбільш загальним поданням функціональних вимог до системи

Суть діаграми прецедентів – проєктована система подається у вигляді набору (графу) сутностей (акторів), які взаємодіють із системою через так звані варіанти використання (use case, прецеденти). При цьому актором (actor) або дійовою особою називається будь-яка сутність, яка взаємодіє із системою ззовні. Це є людина, технічний пристрій, програма або будь-яка інша система, яка є джерелом впливу на модельовану систему так, як визначить сам розробник. У свою чергу,

варіант використання є описом функціональностей, які система надає актору [20, с.38].

Основні актори:

– Користувач: Людина, яка використовує систему для оренди автомобілів;

– Система: Веборієнтована інформаційна система для оренди автомобілів;

– Адміністратор: Людина, яка керує системою, включаючи додавання та видалення автомобілів, та керування користувачами.

Список варіантів використання для користувача:

1) Реєстрація в системі:
   a) Користувач вводить свої дані (ім'я, електронну пошту, пароль);
   b) Система перевіряє валідність введених даних;
   c) Система створює обліковий запис користувача.
2) Авторизація в системі:
   a) Користувач вводить свою електронну пошту та пароль;
   b) Система перевіряє введені дані;
   c) У разі успішної перевірки, система надає користувачеві доступ до особистого кабінету.
3) Перегляд каталогу автівок:
   a) Користувач переходить на сторінку каталогу автівок;
   b) Система відображає список доступних автомобілів;
   c) Користувач застосовує фільтри для пошуку автомобілів певної марки.
4) Додавання автомобіля в улюблені:
   a) Користувач клікає на кнопку "серця" на картці оголошення автомобіля;
   b) Система перевіряє, чи авторизований користувач;
   c) У разі успішної перевірки, система додає автомобіль до списку улюблених.

5) Видалення автомобіля з улюблених:

   a) Користувач клікає на кнопку "серця" на картці оголошення автомобіля (який вже знаходиться в улюблених);

   b) Система перевіряє, чи авторизований користувач;

   c) У разі успішної перевірки, система видаляє автомобіль зі списку улюблених.

6) Перегляд детальної інформації про автомобіль:

   a) Користувач клікає на кнопку "Детальніше" на картці оголошення автомобіля;

   b) Система відображає модальне вікно з детальною інформацією про автомобіль та умови його оренди.

7) Зв'язок з компанією:

   a) Користувач клікає на кнопку "Rental car";

   b) Система надає можливість зв'язатися з компанією за номером телефону.

Список варіантів використання для адміністратора:

– Додавання нового автомобіля;
– Видалення автомобіля;
– Додавання нового користувача;
– Видалення користувача;
– Редагування інформації про автомобіль.

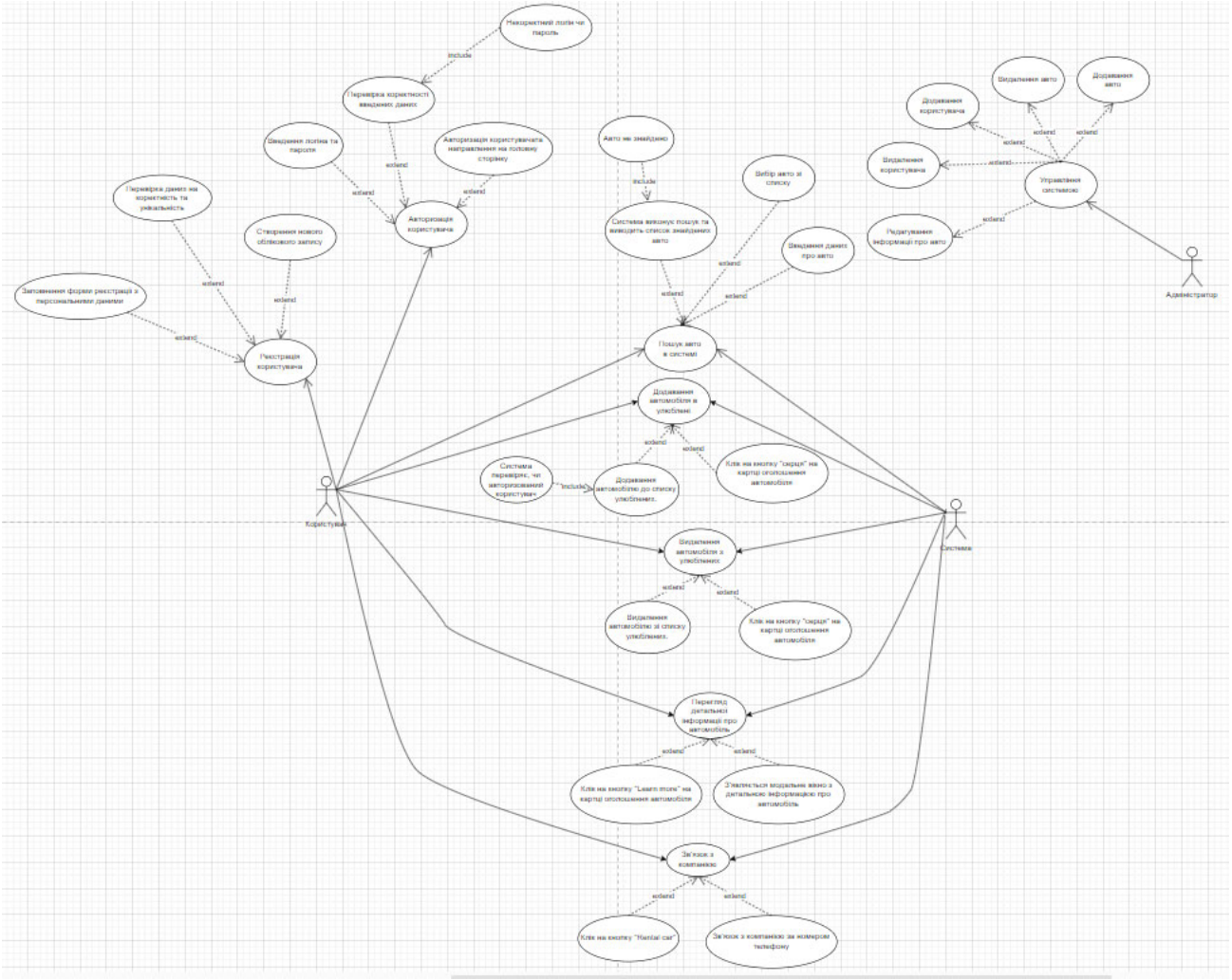

Рисунок 2.4 – UML діаграма: Use Cases

Діаграма класів (class diagram) використовуються для моделювання статичного подання системи з точки зору проєктування в термінології класів об'єктноорієнтованого програмування. Діаграма класів може відбивати різні взаємозв'язки між окремими сутностями предметної області, такими як об'єкти і підсистеми, а також описує їх внутрішню структуру і типи відносин [20, с.45].

Класи

Admin: Цей клас відповідає за керування системою. Він має такі атрибути:

- Car list: Список авто, з яким адміністратор прцює;
- List of users: Список користувачів, з яким адміністратор прцює.

Car: Цей клас описує автомобіль. Він має такі атрибути:

– id: Унікальний ідентифікатор автомобіля;

– year: Рік випуску автомобіля;
– make: Виробник автомобіля;
– model: Модель автомобіля;
– type: Тип автомобіля;
– img: Зображення автомобіля;
– description: Опис автомобіля;
– fuelConsumption: Витрата палива автомобіля;
– engineSize: Об'єм двигуна автомобіля;
– accessories: Аксесуари автомобіля.
– functionalities: Функціональні можливості автомобіля;
– rentalPrice: Ціна оренди автомобіля;
– rentalCompany: Орендна компанія, яка надає автомобіль;
– address: Адреса орендної компанії;
– rentalConditions: Умови оренди автомобіля;
– mileage: Пробіг автомобіля.

User: Цей клас описує користувача системи. Він має такі атрибути:

– id: Унікальний ідентифікатор користувача;
– fullName: Повне ім'я користувача;
– email: Адреса електронної пошти користувача;
– passwordHash: Хеш пароля користувача;
– createdAt: Дата та час створення облікового запису користувача;
– updatedAt: Дата та час останнього оновлення облікового запису користувача.

Favorites: Цей клас описує список улюблених автомобілів користувача. Він має такі атрибути:

– userId: Унікальний ідентифікатор користувача;
– carId: Унікальний ідентифікатор автомобіля.

Відносини

– Admin має багато користувачів: Один адміністратор може керувати багатьма користувачами;

– Admin має багато автомобілів: Один адміністратор може керувати багатьма автомобілями;

– Користувач має багато автомобілів: Один користувач може орендувати багато автомобілів;

– Користувач має багато улюблених автомобілів: Один користувач може додати до списку улюблених багато автомобілів;

– Автомобіль може бути в списку улюблених багатьох користувачів: Один автомобіль може бути доданий до списку улюблених багатьох користувачів.

Методи

Admin:

– addUser(): Додати нового користувача до системи;

– deleteUser(): Видалити користувача з системи;

– addCar(): Додати нову автівку до системи;

– deleteCar(): Видалити автівку із системи;

– getCar(): Отримати інформацію про автомобіль;

– getUser(): Отримати інформацію про користувача.

Car:

– add(): Додати новий автомобіль до системи;

– remove(): Видалити автомобіль з системи;

– getDetails(): Отримати детальну інформацію про автомобіль;

– editDetails(): Редагувати детальну інформацію про автомобіль;

– addFavotites(): Додати автомобіль до списку улюблених;

– remove Favorites(): Видалити автомобіль зі списку улюблених.

User:

– register(): Реєстрація нового користувача у систему;

– login(): Авторизація користувача у систему;

– logout(): Вихід користувача з системи;

– getFavorites(): Отримати інформацію про улюблені автомобілі.

Favorite:

– add(): Додати автомобіль до улюблених;

– remove(): Видалити автомобіль з улюблених.

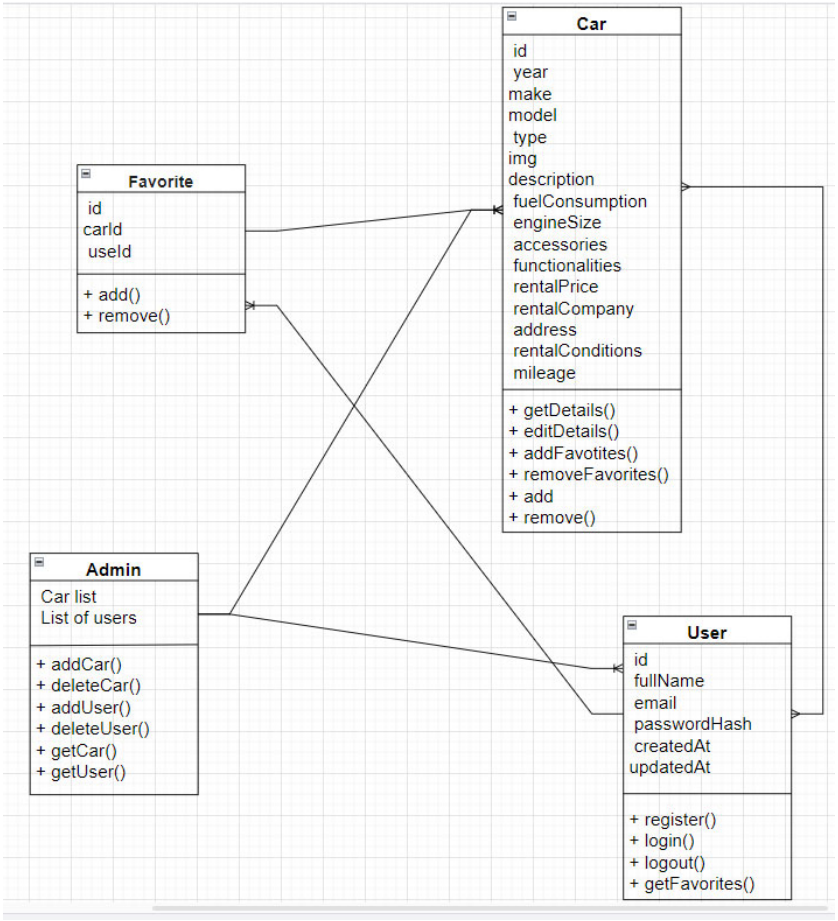

Рисунок 2.5 – UML діаграма: Classes

Діаграми діяльності (Activity diagram) створюються на різних етапах життєвого циклу системи для моделювання послідовності бізнес-процесів або потоку дій, реалізованих методами класів.

Діаграма діяльності показує потік переходів від однієї діяльності до іншої, при чому увага фіксується на наслідках діяльності. Сам результат може призвести до зміни стану системи або повернення деякого значення. На діаграмі діяльності відображається логіка або послідовність переходу від однієї діяльності до іншої. [20, с.57].

Діючі особи:

– Користувач;

– Система оренди автомобілів.

Сценарії:

Головна сторінка:

– Користувач переходить на головну сторінку;
– Система відображає опис послуг, що надає компанія;
– Користувач переходить до сторінки реєстрації;
– Користувач переходить до сторінки авторизації;
– Користувач переходить до каталогу автомобілів;
– Користувач переходить до списку улюблених автомобілів.

Реєстрація:

– Користувач переходить на сторінку реєстрації;
– Користувач вводить свої дані (ім'я, email, пароль);
– Система валідує введені дані;
– Якщо дані валідні, система реєструє нового користувача;
– Користувач перенаправляється на сторінку авторизації.

Авторизація:

– Користувач переходить на сторінку авторизації;
– Користувач вводить свій email та пароль;
– Система валідує введені дані;
– Якщо дані валідні, система авторизує користувача;
– Користувач перенаправляється до каталогу автомобілів.

Каталог автомобілів:

– Користувач переходить до каталогу автомобілів;
– Система відображає список доступних для оренди автомобілів;
– Користувач переглядає детальну інформацію про автомобіль;
– Користувач додає автомобіль до списку улюблених;
– Користувач застосовує фільтр за маркою автомобіля;
– Користувач завантажує більше оголошень;
– Користувач зв'язується з компанією за номером телефону.

Детальна інформація про автомобіль:

– Користувач відкриває модальне вікно з детальною інформацією про автомобіль;

– Система відображає фото, опис, характеристики, умови оренди та контактні дані компанії;

– Користувач закриває модальне вікно з детальною інформацією про автомобіль.

Список улюблених:

– Користувач переходить до списку улюблених;

– Система відображає список автомобілів, доданих до улюблених;

– Користувач видаляє автомобіль зі списку улюблених.

Неіснуючий маршрут:

– Користувач переходить за неіснуючим маршрутом;

– Система перенаправляє користувача на головну сторінку.

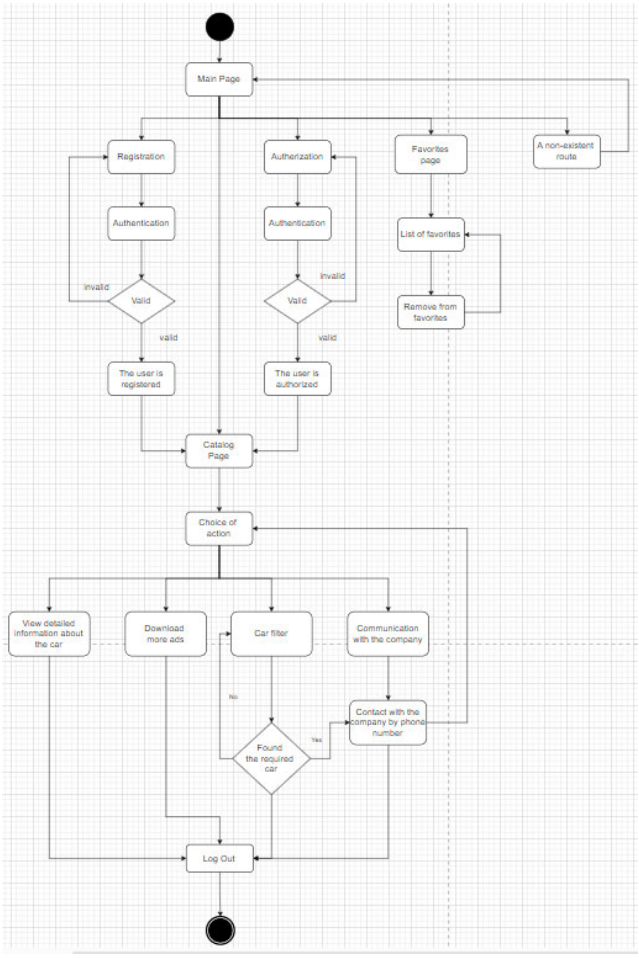

Рисунок 2.6 – UML діаграма: Activity

2.5 Висновки до розділу 2

У процесі проєктування програмного забезпечення для системи оренди автомобілів було розроблено інформаційну модель предметної області, архітектуру програмного застосунку, проєктування бази даних та модель програмної системи.

В процесі аналізу було ідентифіковано основні сутності та їх взаємозв'язки в системі оренди автомобілів. Була розроблений ER-модель, яка відображає ключові сутності, серед яких є користувач, автомобіль, улюблені оголошення, та оренда. Ця модель надає загальне уявлення про структуру та функціональність системи.

Архітектурний підхід до розробки програмного забезпечення передбачає використання клієнт-серверної архітектури з веб-орієнтованим інтерфейсом користувача. Frontend буде розроблений з використанням бібліотеки React та Redux для керування станом додатку, в той час як backend буде побудований з використанням Node.js та MongoDB для зберігання даних.

Структура бази даних розроблена з урахуванням потреб системи в зберіганні та обробці даних про користувачів, автомобілі, оголошення та інші сутності. Була розроблена діаграма DFD потоків даних, яка відображає потоки даних та обробку цих даних в інформаційній системі.

Для моделювання програмної системи було використано UML діаграми: Use Cases, Classes, та Activity. Use Case діаграма дозволила визначити основні функціональні можливості системи та їх взаємозв'язки з користувачами. Діаграма Classes відображає структуру класів системи та їх взаємозв'язки, в той час як діаграма Activity ілюструє послідовність дій у різних сценаріях використання.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Розробка запитів до бази даних відповідно до функціоналу системи

В даному розділі розглядається процес розробки запитів до бази даних для взаємодії з інформаційною системою. Він потрібен для реалізації backend, так як взаємодія з базою даних дозволяє зберігати, зчитувати, оновлювати та видаляти дані про користувачів, автомобілі, оголошення тощо.

Використовується MongoDB для гнучкості у структурі даних та дозволяє легко масштабувати базу даних зростанням обсягу інформації.

Для виконання базових запитів, які представлені на рисунках 3.1 – 3.4, буде використано програму Postman, яка дозволяє тестувати веб-інтерфейси API.

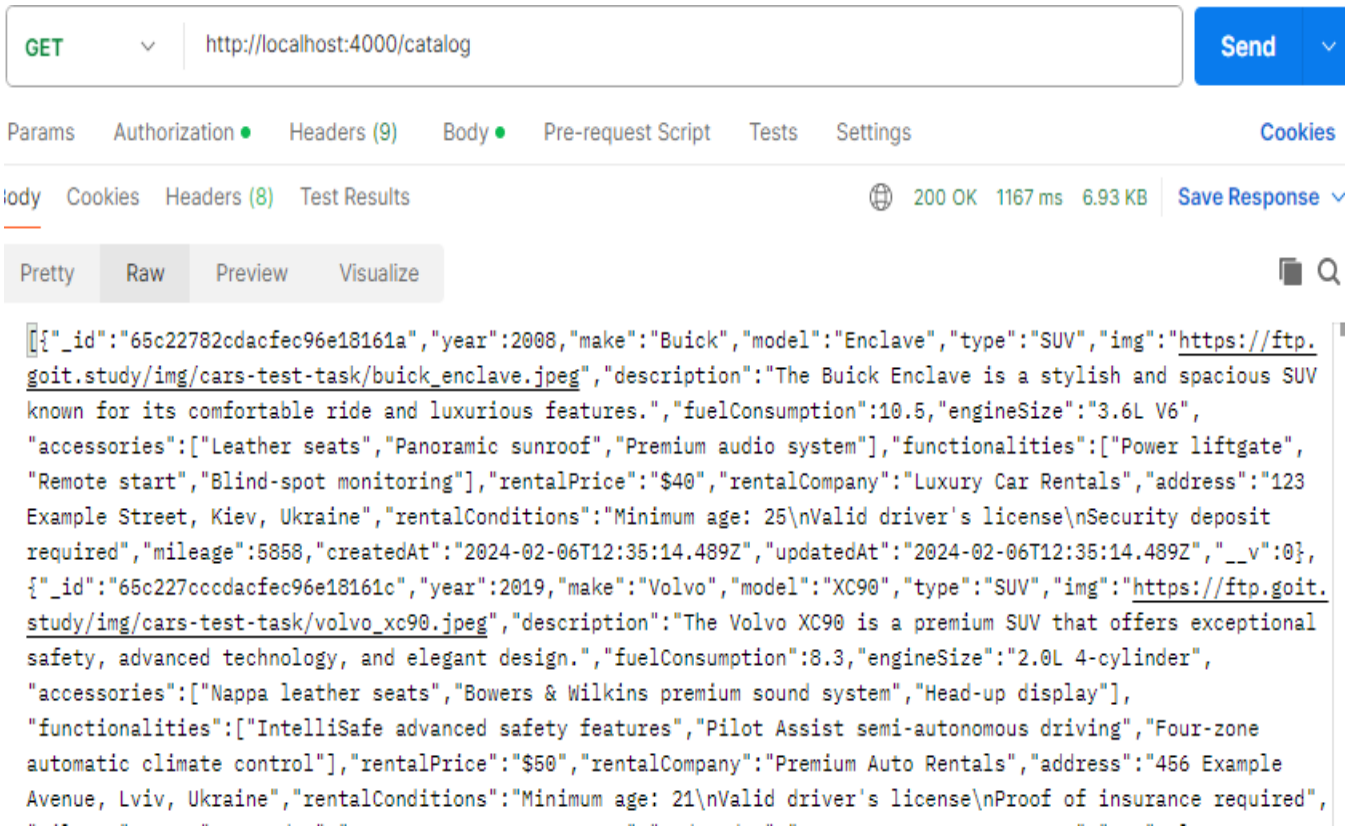

Рисунок 3.1 – Запит на отримання всіх автомобілів

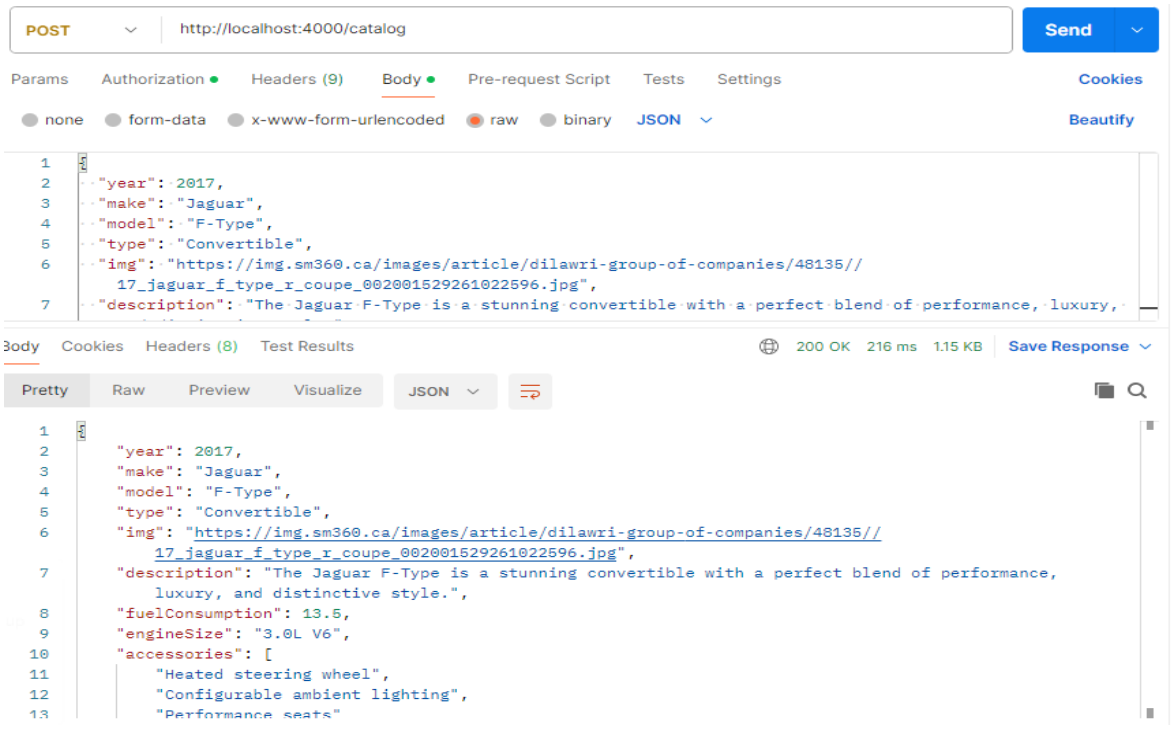

Рисунок 3.2 – Запит на додавання нового автомобіля

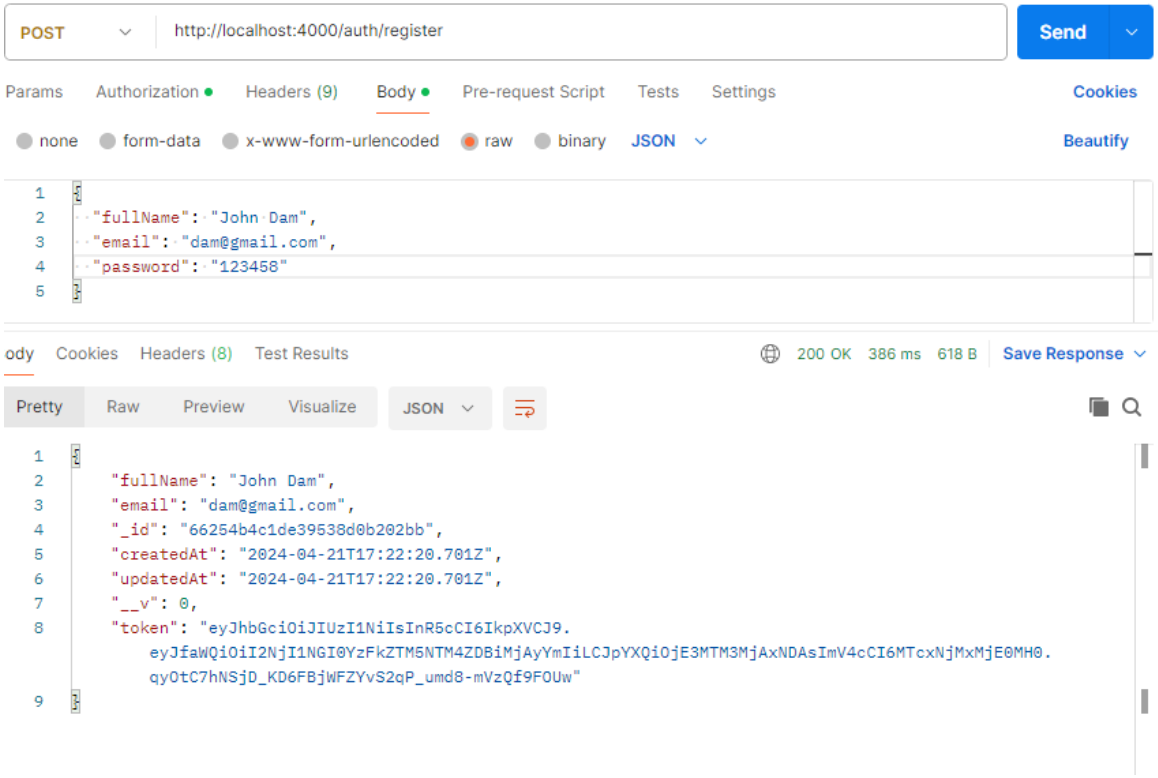

Рисунок 3.3 – Запит на додавання нового користувача

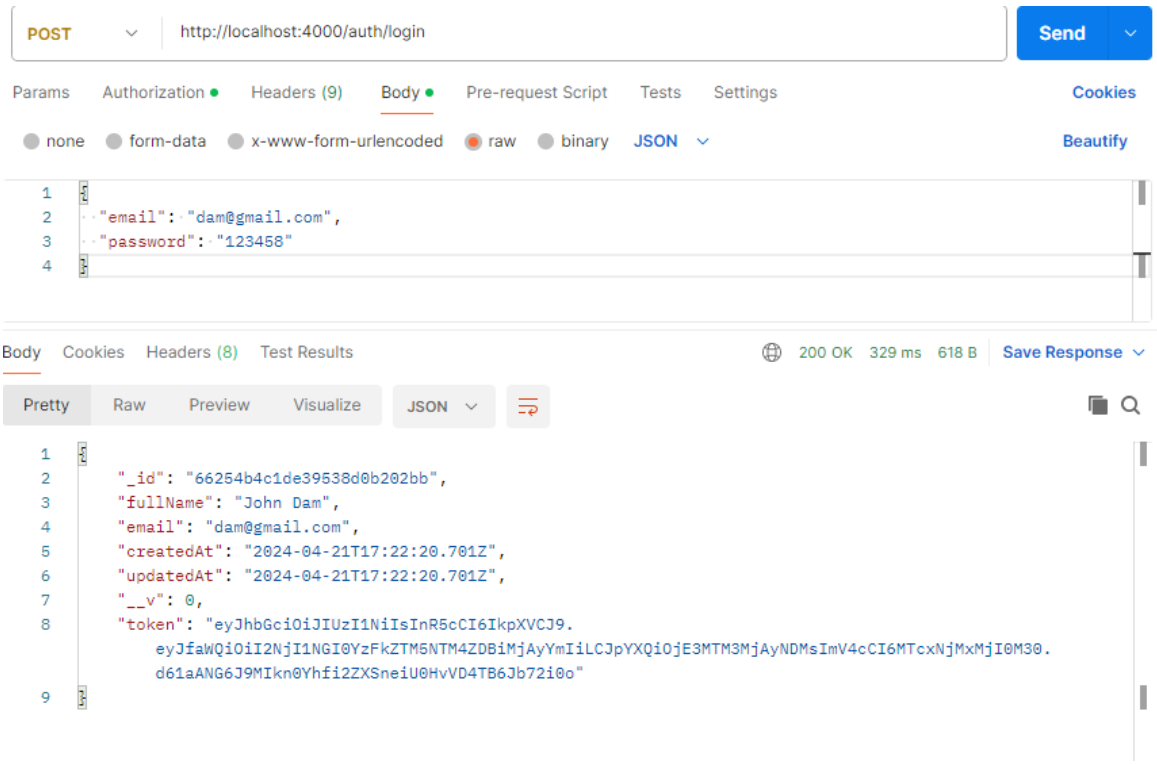

Рисунок 3.4 – Запит на авторизацію користувача

3.2 Розробка серверної частини з використанням Express.js та Node.js

У даному розділі розглядається процес розробки backend. Він відповідає за обробку запитів, взаємодію з базою даних, аутентифікацію, регістрацію та авторизацію користувачів, а також надання необхідної інформації frontend.

В процесі розробки системи, було обрано технологію Express.js разом з базою даних MongoDB для забезпечення швидкої та ефективної роботи з колекцією даних. Express.js дозволяє швидко створювати масштабовані веб-додатки з допомогою простого API. MongoDB використовується як база даних, завдяки гнучкого та швидкого доступу до даних.

Серверна частина веб-ситеми розділена на кілька модулів та контролерів для кращої організації та збереження чистоти коду. Основні модулі та контролери включають у себе реєстрацію та авторизацію користувачів, роботу з каталогом автомобілів та валідацію даних.

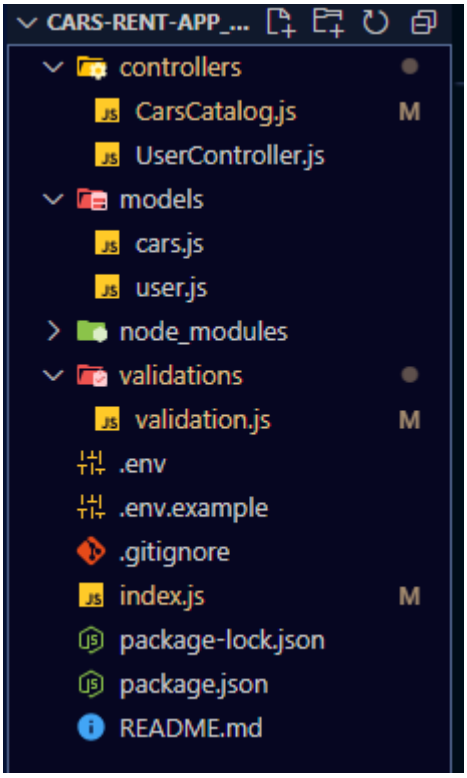

Рисунок 3.5 – Структура папок backend коду

user.js:

У цьому файлі є код, який визначає схему користувача для MongoDB за допомогою бібліотеки Mongoose:

UserSchema: Це об'єкт, який описує схему користувача. Він містить поля, які визначаються для кожного користувача:

– fullName: Поле для повного імені користувача. Тип даних - рядок. Встановлено властивість required: true, щоб забезпечити, що дані про ім'я завжди мають бути введені при створенні користувача;

– email: Поле для електронної пошти користувача. Тип даних - рядок. Встановлено required: true і unique: true, щоб гарантувати, що кожна електронна адреса унікальна і обов'язкова для введення;

– passwordHash: Поле для хешу пароля користувача. Тип даних - рядок. Встановлено required: true, оскільки хеш пароля також обов'язковий для збереження;

– timestamps: Ця опція встановлює автоматичне створення двох додаткових полів для кожного запису користувача: createdAt і updatedAt. Поля цієї пари містять дату та час створення та останнього оновлення відповідного запису.

cars.js:

У цьому файлі є код, який визначає схему для об'єктів автомобілів у базі даних MongoDB за допомогою бібліотеки Mongoose:

CarSchema: Це об'єкт, який описує схему автомобіля. Він містить поля, які визначаються для кожного автомобіля:

– year: Рік випуску автомобіля. Тип даних - число. Встановлено властивості required: true для забезпечення того, що рік завжди вказується, а також min та max для обмеження значень року в межах від 1900 до поточного року;

– make: Виробник автомобіля. Тип даних - рядок. Обов'язкове для введення;

– model: Модель автомобіля. Тип даних - рядок. Обов'язкове для введення;

– type: Тип автомобіля. Тип даних - рядок. Обов'язкове для введення;

– img: Посилання на зображення автомобіля. Тип даних - рядок. Обов'язкове для введення. Встановлено валідацію, щоб перевірити, чи є значення посиланням на дійсну URL-адресу;

– description: Опис автомобіля. Тип даних - рядок. Обов'язкове для введення;

– fuelConsumption: Споживання пального. Тип даних - число. Обов'язкове для введення;

– engineSize: Об'єм двигуна. Тип даних - рядок. Обов'язкове для введення;

– accessories: Масив додаткового обладнання. Тип даних - масив. Обов'язкове для введення;

– functionalities: Масив функціональностей. Тип даних - масив. Обов'язкове для введення;

– rentalPrice: Ціна оренди. Тип даних - рядок. Обов'язкове для введення;

– rentalCompany: Назва компанії, що здає в оренду автомобіль. Тип даних - рядок. Обов'язкове для введення;

– address: Адреса розташування автомобіля. Тип даних - рядок. Обов'язкове для введення;

– rentalConditions: Умови оренди автомобіля. Тип даних - рядок. Обов'язкове для введення;

– mileage: Пробіг автомобіля. Тип даних - число. Обов'язкове для введення. Встановлено мінімальне значення пробігу як 0;

– timestamps: Ця опція встановлює автоматичне створення двох додаткових полів для кожного запису автомобіля: createdAt і updatedAt. Поля цієї пари містять дату та час створення та останнього оновлення відповідного запису.

UserController.js:

У цьому файлі є код, який виконує реєстрацію та вхід користувачів у систему за допомогою JWT (JSON Web Token) для автентифікації:

– register: Ця функція обробляє запит на реєстрацію нового користувача. Вона перевіряє, чи є помилки у валідації даних, які надійшли від клієнта. Якщо так, повертає помилку зі статусом 400 та переліком помилок. Після цього вона хешує пароль користувача за допомогою bcrypt, створює об'єкт користувача з даними, які отримано від клієнта, зберігає його в базі даних та повертає JWT-токен разом із даними користувача без хешу пароля;

– login: Ця функція обробляє запит на вхід користувача. Вона знаходить користувача за його email, перевіряє, чи введений пароль вірний, порівнюючи хеш пароля в базі даних з хешем введеного пароля за допомогою bcrypt. Якщо вірно, генерує JWT-токен та повертає його разом із даними користувача без хешу пароля. Якщо користувача не знайдено або пароль невірний, повертається відповідна помилка.

CarsCatalog.js:

У цьому файлі є код, який відповідає за створення та отримання інформації про автомобілі:

– create: Ця функція використовується для створення нового запису про автомобіль у базі даних. Вона отримує дані про автомобіль з запиту, створює новий об'єкт Car з цими даними та зберігає його в базі даних. Після успішного збереження функція повертає створений об'єкт автомобіля у відповідь;

– getAll: Ця функція використовується для отримання списку автомобілів з бази даних. Вона приймає параметри запиту, такі як сторінка, кількість елементів на сторінці, бренд автомобіля, ціновий діапазон, діапазон пробігу тощо. На основі цих параметрів функція формує фільтр для пошуку автомобілів у базі даних та виконує запит до бази даних. Після цього вона повертає список знайдених автомобілів у відповідь.

Обидва обробники створення та отримання автомобілів повертають JSON-відповідь з даними автомобілями у разі успішної операції або відповідну помилку з статусом 500 у разі невдачі.

validation.js:

У цьому файлі є код, який визначає набори правил валідації за допомогою бібліотеки express-validator для різних ендпоінтів вашого додатку Express. Ось короткий опис кожного з них:

1) registerValidation: Цей набір правил валідації використовується для перевірки даних, які надходять при реєстрації нового користувача. Він містить наступні правила:

   a) email: Перевіряє, чи введена коректна електронна пошта;

   b) password: Перевіряє, чи пароль має принаймні 6 символів;

   c) fullName: Перевіряє, чи введено повне ім'я користувача (мінімум 3 символи).

2) loginValidation: Цей набір правил валідації використовується для перевірки даних, які надходять при вході користувача. Він містить ті ж правила, що і registerValidation;

3) carsCatalogValidation: Цей набір правил валідації використовується для перевірки даних, які надходять при створенні нового запису про автомобіль. Він містить наступні правила:

a) Перевірка року випуску автомобіля на те, щоб він був у встановленому діапазоні;

b) Перевірка типів даних для рядків та чисел у різних полях;

c) Перевірка URL-адреси зображення;

d) Перевірка, що деякі поля не порожні (наприклад, make, model, description тощо);

e) Перевірка, що деякі поля є масивами (наприклад, accessories, functionalities);

f) Перевірка діапазону значень для mileage.

Кожне правило включає повідомлення про помилку, яке буде повернуто, якщо валідація не пройде успішно.

index.js:

– Цей код визначає сервер за допомогою фреймворку Express, який взаємодіє з базою даних MongoDB за допомогою бібліотеки Mongoose. Ось короткий опис кожної частини:

– dotenv.config(): Цей рядок завантажує змінні середовища з файлу .env у змінній process.env;

– mongoose.connect(DB_HOST): Підключення до бази даних MongoDB за допомогою зазначеної у змінній DB_HOST URL;

– app: Створення нового екземпляру додатку Express;

– app.use(express.json()): Використання middleware для розбору тіла запиту у форматі JSON;

– app.use(cors()): Використання middleware CORS для дозволу перетину ресурсів між доменами;

– app.post('/auth/register', registerValidation, register): Реєстрація маршруту для обробки POST-запитів на реєстрацію користувачів. При обробці запиту виконується перевірка валідації з допомогою registerValidation, а потім виконується функція обробника register;

– app.post('/auth/login', loginValidation, login): Реєстрація маршруту для обробки POST-запитів на вхід користувачів. При обробці запиту виконується

перевірка валідації з допомогою loginValidation, а потім виконується функція обробника login;

– app.post('/catalog', carsCatalogValidation, create): Реєстрація маршруту для обробки POST-запитів на створення нового запису автомобіля в каталозі. При обробці запиту виконується перевірка валідації з допомогою carsCatalogValidation, а потім виконується функція обробника create;

– app.get('/catalog', getAll): Реєстрація маршруту для обробки GET-запитів на отримання всього каталогу автомобілів. При обробці запиту виконується функція обробника getAll;

– app.listen(PORT, (err) => { ... }): Запуск сервера на вказаному порту PORT. Якщо виникає помилка, вона виводиться у консоль. Інакше виводиться повідомлення про успішний запуск сервера.

Розробка серверної частини веб-додатка для оренди автомобілів з використанням Express.js та Node.js дозволяє забезпечити ефективну та стабільну роботу системи.

3.3 Розробка клієнтської частини засобами React

Розробка клієнтської частини веборієнтованої системи для оренди автомобілів, за допомогою засобів React - це один із ключових етапів проекту. React - це потужний інструмент для створення інтерактивних та ефективних користувацьких інтерфейсів, що надає зручний підхід до розробки складних додатків. У даному пункті розглядається структура, компоненти та функціонал клієнтської частини.

Для веборієнтованої системи для оренди автомобілів структура з папками components, pages, redux, та services.

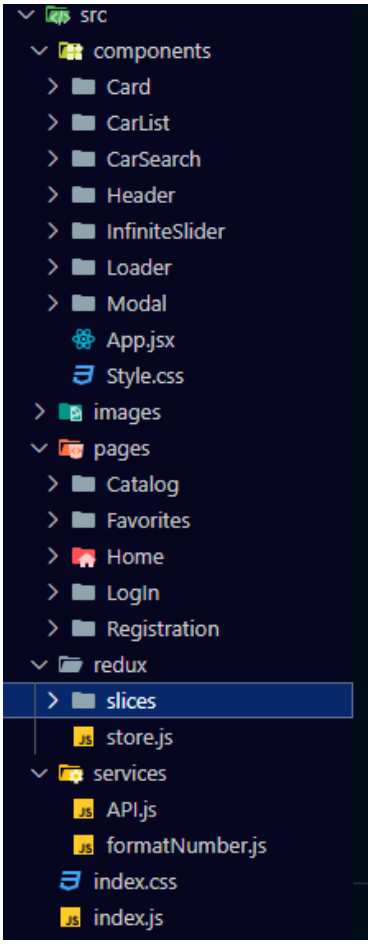

Рисунок 3.6 – Структура папок frontend коду

Папка components:

У папці components містяться незалежні компоненти, які являють собою частини функціоналу для сторінок ситеми. У папці pages розташовані компоненти, які відповідають окремим сторінкам додатку. Папка redux містить файли, пов’язані з реалізацією стану за допомогою Redux. Вона містить редуктори для авторизації, каталогу та улюблених оголошень.. У папці services знаходиться файли для роботи з віддаленим сервером та API.

– Компонент Card відображає оголошення автомобіля. Він використовує стан useState, щоб відслідковувати відкриття модального вікна. Хуки useDispatch і useSelector з бібліотеки React Redux потрібні для взаємодії зі store Redux-у. Є можливість додавати автомобілі до обраних або видаляти їх з обраних. Це залежить від статусу авторизації користувача. Відображає інформацію про автомобіль, таку як адреса, компанія з прокату, модель, пробіг, функціональності та інші

характеристики. Кнопка "Learn more" відкриває модальне вікно з додатковою інформацією про автомобіль;

– Компонент CarList відображає галерею автомобілів з використанням React. Він отримує список автомобілів зі зберігання Redux, фільтрує їх згідно встановленим фільтрам і відображає лише ті, які відповідають фільтру. Кожен автомобіль відображається за допомогою компонента CarCard. useEffect використовується для отримання каталогу автомобілів після завантаження компонента або при зміні фільтра;

– Компонент CarSearch відображає пошук автомобілів у системі за допомогою React. Він містить текстові поля для введення бренду автомобіля, діапазону цін та діапазону пробігу. Кнопка "Search" викликає функцію onSearch, яка передає введені дані для пошуку. Кнопка "Reset" скидає введені дані та викликає функцію onReset;

– Компонент Header відображає блок в верхній частині сайту. Він містить навігаційне меню з посиланнями на головну сторінку, каталог та обране. Якщо користувач автентифікований (isAuth = true), він може вийти з системи за допомогою кнопки "Log out". Якщо користувач не автентифікований, він може перейти на сторінки реєстрації та входу;

– Компонент InfiniteSlider відображає Slider у React, який відображає зображення з масиву images. Компонент використовує хук useState для збереження поточного індексу зображення та useEffect для автоматичного перемикання зображень кожні 4 секунди. При завантаженні компоненту useEffect встановлює інтервал, який змінює індекс зображення, інкрементуючи його, поки не досягне останнього елементу масиву, після чого повертається до першого елементу. У useEffect також вказано, що потрібно очищати інтервал при розмінюванні компоненту;

– Компонент Loader відображає Loader, який використовується для відображення анімації завантаження. Він імпортує компонент Dna з бібліотеки "react-loader-spinner" і використовує його для створення анімації;

– Компонент Modal відображає модальне вікно для відображення інформації про автомобіль. Він приймає два пропи: car (об'єкт з інформацією про автомобіль) та onCloseModal (функція для закриття модального вікна). Використовує useCallback, useEffect, createPortal, а також стилізовані компоненти для створення власного дизайну модального вікна. Інформація про автомобіль, передана через car проп, використовується для відображення заголовку, зображення, опису, додаткових аксесуарів, функціоналів та умов оренди. Використовуються події клавіатури та миші для закриття модального вікна. Завершується створення модального вікна, яке потім буде відображено в modalRoot, який шукається в DOM за допомогою querySelector;

– Компонент App розділяє сторінки за допомогою маршрутів, дозволяючи користувачеві переходити між різними частинами додатка. Код також використовує "ліниве завантаження" (lazy loading), щоб зменшити час завантаження сторінок. Хук useState використовується для створення стану для зберігання списку автомобілів. Функція favoriteToggle змінює статус улюблених автомобілів та зберігає їх у локальному сховищі. Використовується Suspense, який показує компонент <Loader /> під час завантаження сторінок.Маршрути, такі як /, /registration, /login, /catalog, /favorites, відображають відповідні сторінки.

Папка pages:

– Файл Catalog відображає каталог автомобілів. Він приймає дві функції як властивості: setCars (для встановлення автомобілів) та favoriteToggle (для перемикання улюблених автомобілів). Використовує хуки useState та useEffect для управління станом та побічними ефектами. За допомогою Redux здійснюється взаємодія зі станом додатка, а само, з каталогом автомобілів та фільтрами. Відображає список автомобілів (CarList) з можливістю фільтрації (CarSearch), завантаження додаткових автомобілів та навігації по сторінках. Є можливість пошуку автомобілів за допомогою фільтрів та перехід до конкретної сторінки за допомогою форми;

– Файл Favorites відображає список улюблених автомобілів користувача. Він перевіряє, чи авторизований користувач, і відображає повідомлення про

необхідність входу або список улюблених автомобілів, якщо користувач авторизований і вибрав якісь автомобілі;

– Файл Home відображає домашню сторінку системи. Він включає заголовок, дві секції з текстом і зображеннями, а також компонент слайдера для показу фотографій;

– Файл Registration відображає сторінку реєстрації користувача. Користується бібліотекою react-hook-form для управління формою та її валідацією. При відправці форми дані передаються на сервер через функцію fetchRegister. Якщо реєстрація успішна, токен зберігається в локальному сховищі браузера. Якщо користувач вже залогінений (має активну сесію), він буде перенаправлений на головну сторінку;

– Файл LogIn відображає сторінки входу в систему. Він використовує бібліотеку react-hook-form для керування формою вводу. Коли користувач вводить свої дані та натискає кнопку входу, викликається функція onSubmit, яка відправляє дані для аутентифікації через Redux. Якщо аутентифікація успішна, токен зберігається в локальному сховищі. Якщо користувач вже аутентифікований, він автоматично перенаправляється на головну сторінку.

Папка redux:

– auth.js: Імпортується createSlice та createAsyncThunk з Redux Toolkit, а також Axios для роботи з HTTP-запитами. Створюються дві асинхронні функції fetchAuth та fetchRegister, які виконують відповідно авторизацію та реєстрацію користувачів. Визначається початковий стан для авторизації, який містить дані про користувача та його статус. Створюється reducer authSlice, який обробляє результати асинхронних операцій та дозволяє вийти з облікового запису;

– catalog.js: Імпортується createSlice та createAsyncThunk з Redux Toolkit, а також Axios. Створюється асинхронна функція fetchCatalog, яка отримує каталог товарів. Визначається початковий стан для каталогу товарів. Створюється reducer catalogSlice, який обробляє результати отримання каталогу та дозволяє керувати сторінкою товарів;

– favorites.js: Визначається початковий стан для вибраних товарів користувача. Створюється reducer favoritesSlice, який додає та видаляє товари з обраних і зберігає їх у локальному сховищі;

– store.js: Створюється Redux store, який об'єднує редуктори для авторизації, каталогу, обраних товарів та фільтрів.

Папка services:

– API.js: Використовує бібліотеку Axios для створення і налаштування інстанції, яка здійснює HTTP-запити на сервер. Задає основну адреса BASE_URL, яка вказує на локальний сервер. Створює інстанція Axios з базовою адресою сервера. Встановлюється перехоплювач (interceptor) запитів для додавання параметрів до запиту перед відправленням. Визначається функція fetchAPI, яка виконує GET-запит до /catalog на сервері та повертає дані з відповіді. Додає перехоплювач запитів для додавання заголовка Authorization з токеном, який зберігається у локальному сховищі браузера;

– formatNumber.js: Експортує функцію formatNumber, яка приймає число та повертає його рядкове представлення з розділенням розрядів комами.

3.4 Тестування та оцінка якості системи

У цьому пункті проводиться тестування розробленої системи з метою перевірки її функціональності. В даному випадку тестування уявляє собою ручну перевірку функціоналу системи.

Тест реєстрації нового користувача:

При введенні некоректних даних виникає помилка (рис. 3.7). Після успішної реєстрації, користувач перейде на головну сторінку. У localstorage зберігається токен користувача (рис. 3.8).

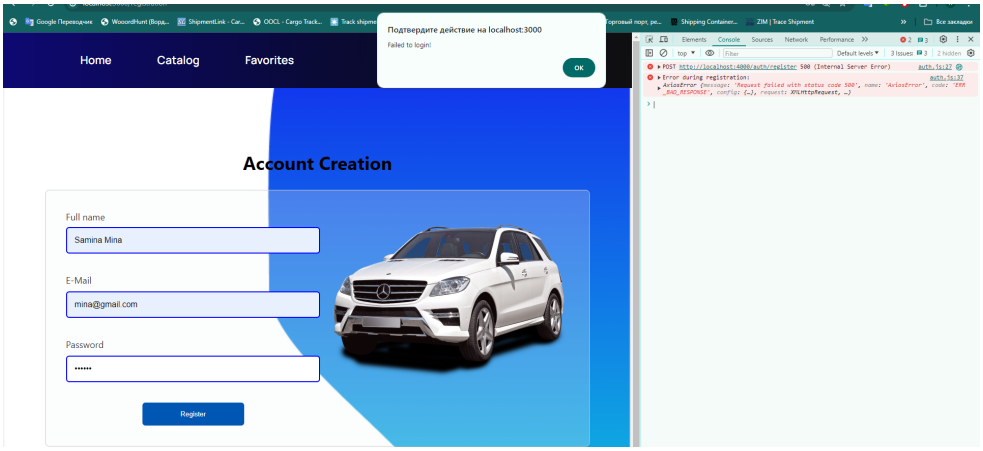

Рисунок 3.7 – Невдала реєстрація користувача

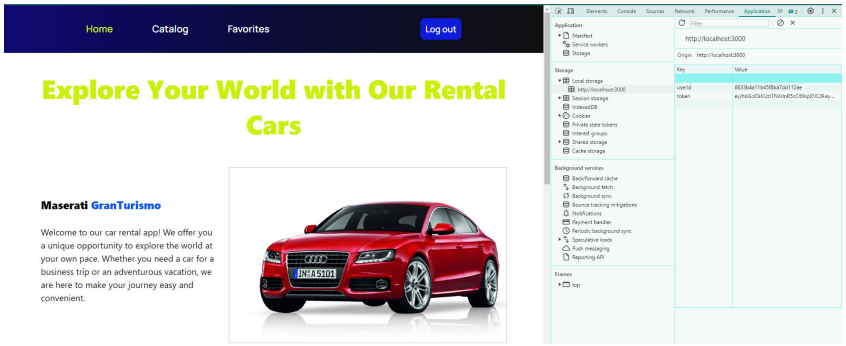

Рисунок 3.8 – Вдала реєстрація користувача

Тест авторизації користувача:

При введенні некоректних даних виникає помилка (рис. 3.9). Після успішної авторизації, користувач перейде на головну сторінку. У localstorage зберігається токен користувача (рис. 3.10).

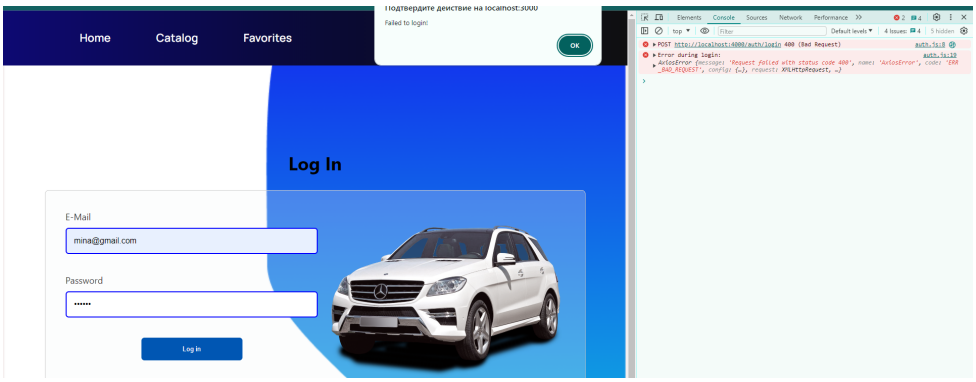

Рисунок 3.9 – Невдала авторизація користувача

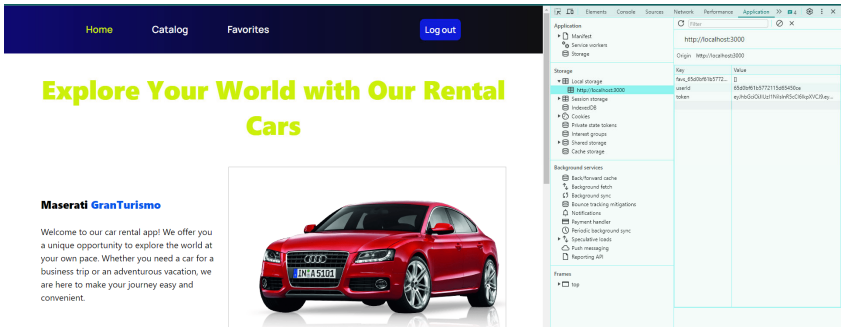

Рисунок 3.10 – Вдала авторизація користувача

Тест перегляду каталогу автомобілів:

Усі наявні автомобілі у каталозі з’являються на сторінці (рис. 3.11). При кліку на кнопку "Learn more" з’являється детальна інформація про автомобіль (рис. 3.12). Фільтрації автомобілів за маркою, ціною або пробігом успішно працює (рис. 3.13).

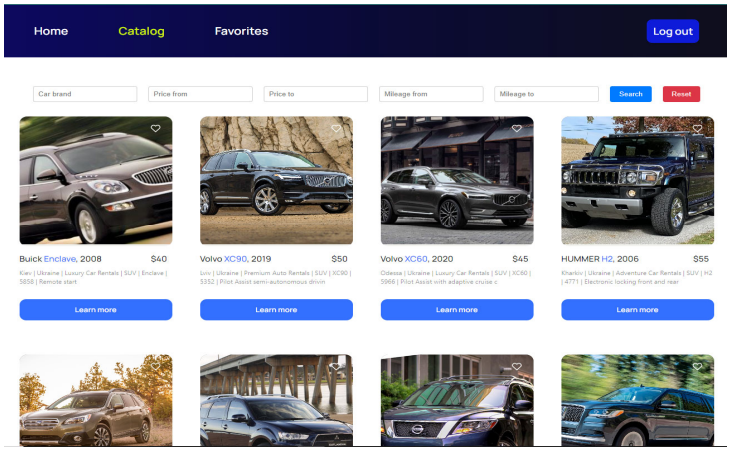

Рисунок 3.11 – Відображення каталогу автомобілів

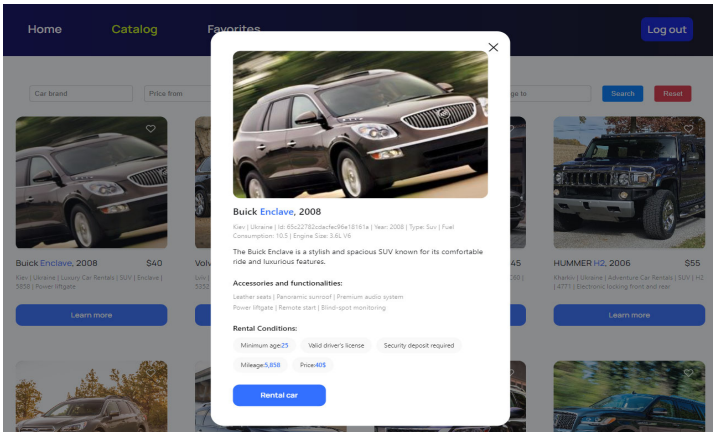

Рисунок 3.12 – З’явлення детальної інформації про автомобіль

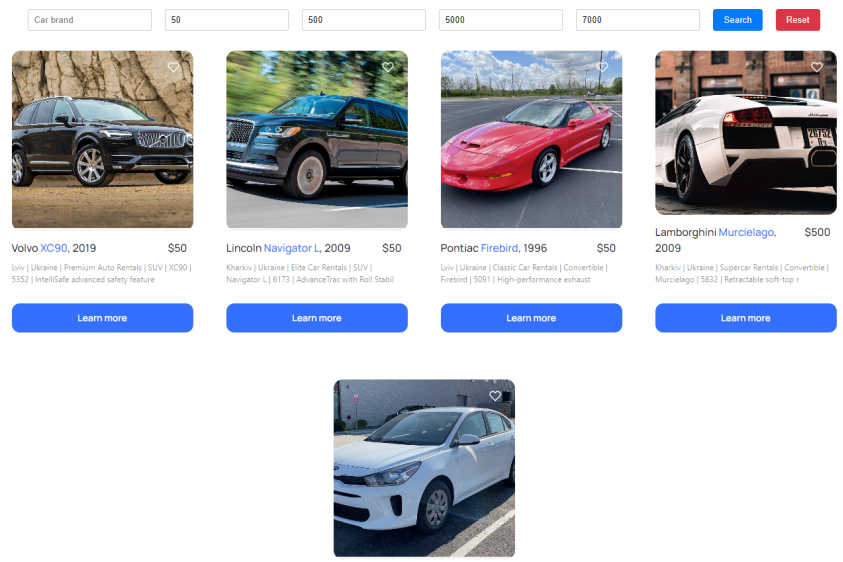

Рисунок 3.13 – Фільтрація автомобілів за обраними параметрами

Тест додавання до улюблених та перегляд:

Авторизований користувач додає до списку улюблених (рис. 3.14). Обрані оголошення відображаються на сторінці "Favorites" . Дані зберігаються у localstorage у масиві "favs" з індексом користувача, для контролю унікальності збережених оголошень у кожного користувача (рис. 3.15). Користувач видаляє оголошення з "Favorites" (рис. 3.16).

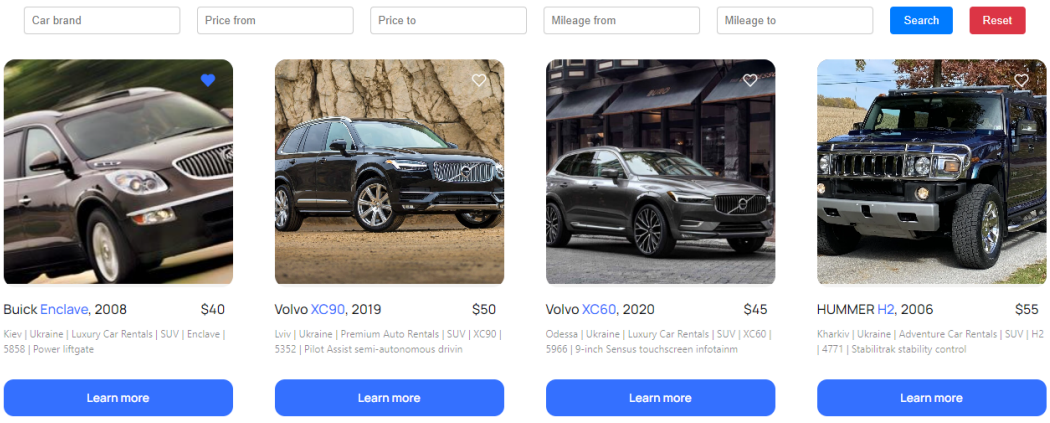

Рисунок 3.14 – Додавання оголошення у "Favorites"

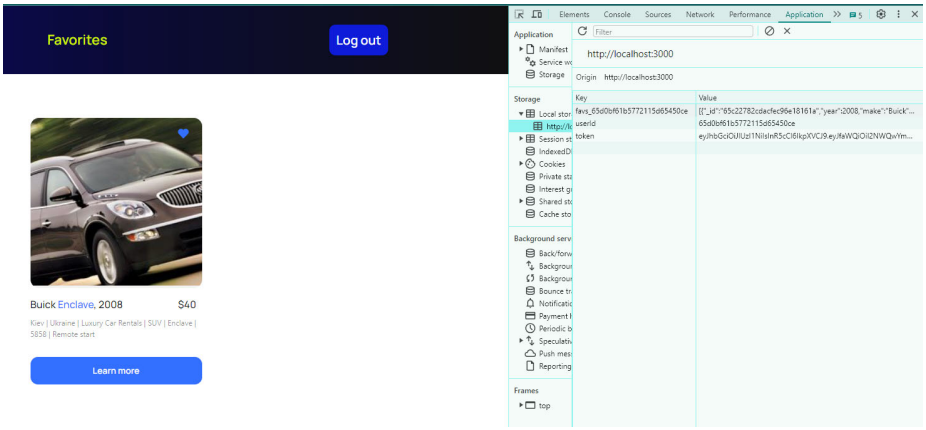

Рисунок 3.15 – Відображення оголошення у "Favorites" з даними у localstorage

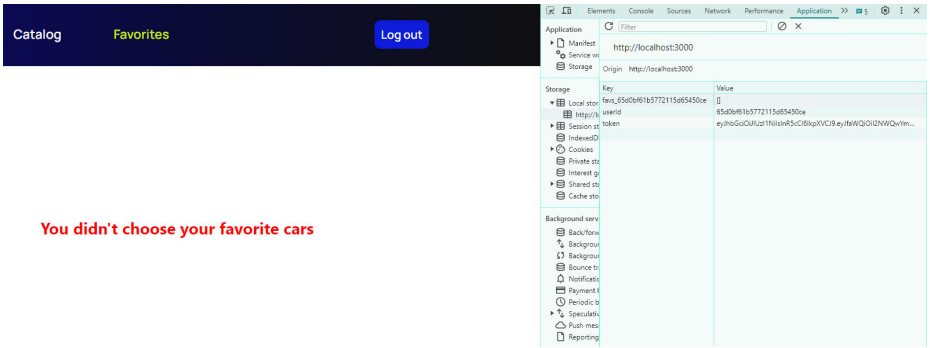

Рисунок 3.16 – Видалення оголошення з "Favorites"

Тест навігації по системі:

Навігаційне меню дозволяє переміщатись між різними сторінками системи. Кнопка "Load more" загружає наступну сторінку оголошень (рис. 3.17), а кнопка "Go back" повертає на попередню сторінку (рис. 3.18). У текстовому полі вноситься номер сторінки, завдяки якому, користувач буде перенесений на обрану сторінку (рис. 3.19).

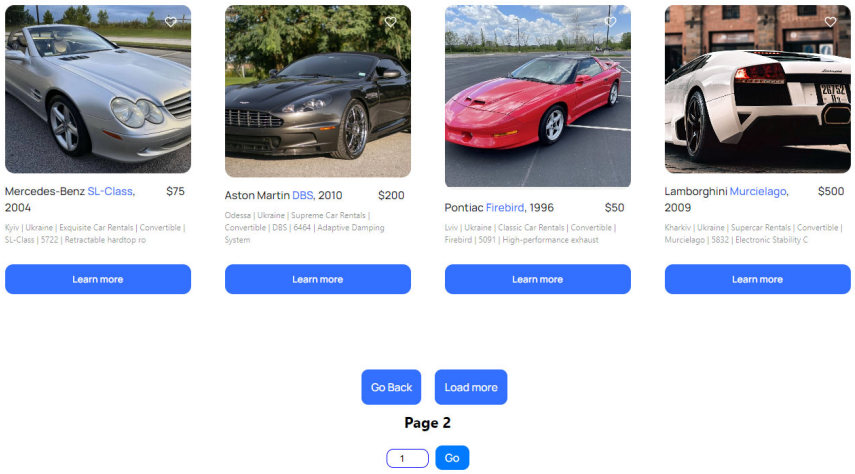

Рисунок 3.17 – Клік на кнопку "Load more"

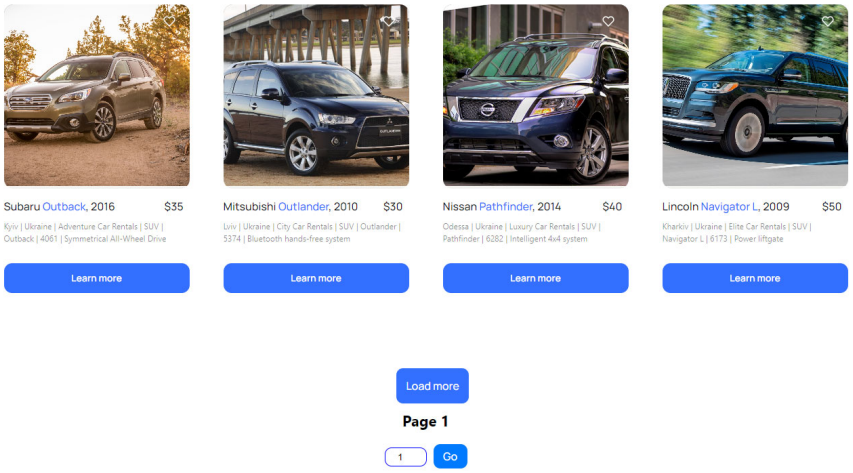

Рисунок 3.18 – Клік на кнопку "Go back"

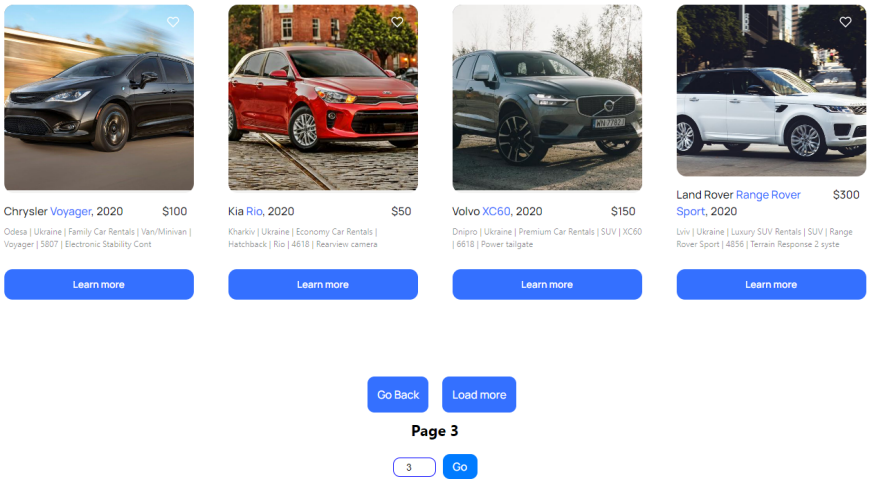

Рисунок 3.19 – Перехід на обрану сторінку

Тест зв'язку з компанією:

За допомогою кнопки "Rental car" можно зв'язатись з компанією для оренди автомобіля (рис. 3.20).

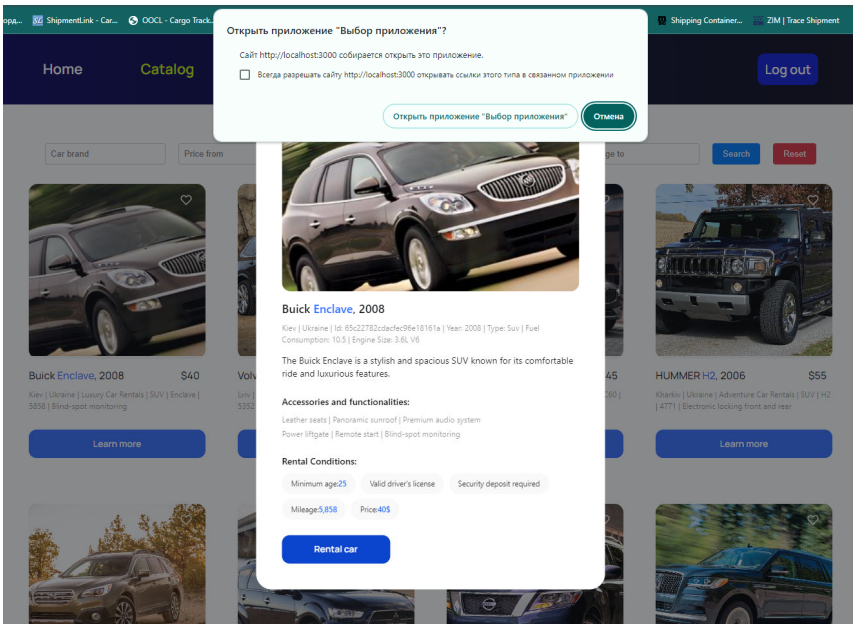

Рисунок 3.20 – Зв’язок з компанією

3.5 Висновки до розділу 3

Під час написання цього пункту були описані ключові аспекти для розуміння розробленої веборієнтованої інформаційної системи:

Для backend використовуються Express.js та Node.js, для створення надійного серверу для обробки запитів взіємодії з автомобільним каталогом, реєстрацією та авторизацією користувача. Використання MongoDB дозволяє зберігати та отримувати дані про користувачів та автомобілі.

Були розроблені модулі для взаємодії з базою даних MongoDB з використанням Express.js та Node.js.

Клієнтська частина була розроблена з використанням бібліотеки React. React Router дозволив ефективно організувати навігацію в системі, а Redux допоміг керувати станом системи.

Протягом тестування було виявлено та виправлено деякі незначні помилки, що дозволило забезпечити стабільну роботу системи. Тестування також допомогло підтвердити правильність роботи функціоналу та забезпечити стабільну роботу ситеми.

Загалом, розробка веборієнтованої ситеми для оренди автомобілів була успішною, а отримані результати відповідають вимогам та очікуванням користувачів.

ВИСНОВКИ

В ході дослідження та розробки веборієнтованої системи для оренди автомобілів було виконано ряд ключових завдань, що дозволяють забезпечити якісне та зручне користування системою.

При аналізі предметної області, було постановлене завдання створення системи, яка забезпечує можливість реєстрації та авторизації користувачів, перегляду каталогу автомобілів, фільтрацію автомобілів за певними параметрами, додавання авто в улюблені, а також надававання інформації про кожен автомобіль та умови його оренди.

Вибір засобів розробки були обрані мова програмування JavaScript, з використанням бібліотеки React для створення зрозумілого і мінімалістичного дизайну, Redux для керування станом додатку та для frontend та Redux для навігації між сторінками додатку. Для розробки для ефективної та зручної серверної частини були використані Node.js та MongoDB.

Проектування програмного продукту включало створення інформаційної моделі предметної області, архітектуру застосунку, проектування бази даних та моделювання програмної системи з використанням UML діаграм.

Розробка системи включала створення серверної частини за допомогою Express.js та Node.js, клієнтської частини засобами React, Redux та React Router, а також тестування ситеми.

У результаті виконаної роботи була розроблена ефективна та зручна веборієнтована ситема для оренди автомобілів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стоянов А.І. Створення веб-водатків з використанням технологій MERN. *Інформаційне суспільство: проблеми та перспективи*: матер. IX міжнар. наук.-практ. конф. Одеса, 24 травня 2024 (депановано).

2. Leonor Teixeira (University of Aveiro, Portugal), Carlos Ferreira (University of Aveiro.“Encyclopedia of Healthcare Information Systems”.Chapter: Web-Enabled System Design for Managing Clinical Information: https://www.igi-global.com/dictionary/web-enabled-system-design-managing/32413#

3. Published in: Proceedings. 13th International Workshop on Database and Expert Systems Applications. Features of a Web-oriented information system: https://ieeexplore.ieee.org/document/1045875

4. Сервіс з онлайн-прокату автомобілів rentcars.com: https://www.rentcars.com/en/

5. Сервіс з онлайн-прокату автомобілів car-rent.ua: https://car-rent.ua/

6. Сервіс з онлайн-прокату автомобілів rental.ua: https://rental.ua/

7. MongoDB – Official website: https://www.mongodb.com/docs/manual/#what-is-mongodb-

8. Реалізація підтримки MongoDB для NestJS Boilerplate з використанням гексагональної архітектури: https://dou.ua/forums/topic/47103/

9. Node.js – Official website: https://nodejs.org/en/

10. Express.js – Official website: https://expressjs.com/uk/

11. Віддалене виконання коду в JsonWebToken: https://corewin.ua/blog/jwt-analysis/

12. Вікіпедія – Bcrypt: https://uk.wikipedia.org/wiki/Bcrypt/

13. Використання CORS в Next.js для обробки cross-origin запитів: https://medium.com/@dmitry.sytnyk/

14. React.js – Official Website: https://react.dev/learn/describing-the-ui

15. Що таке React JS? Як почати вивчати Реакт?: https://cases.media/en/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer

16. Для чого і коли використовується Redux: https://foxminded.ua/shcho-take-redux/

17. Основи React Router: http://fpm.dnu.dp.ua/2019/12/04/react-router/

18. Вікіпедія – Оптимізація для пошукових систем: https://uk.wikipedia.org/wiki/ Оптимізація_для_пошукових_систем

19. Вікіпедія – Маркетинг у соціальних мережах: https://uk.wikipedia.org/wiki/ Маркетинг_у_соціальних_мережах

20. Моделювання програмного забезпечення : навч.-метод. посіб. [Електронний ресурс] / уклад.: С. Ю. Манаков, О. Г. Трофименко, Ю. Г. Лобода, А. І. Дика ; Нац. ун-т «Одеська юрид. академія». – Одеса : Фенікс, 2023. – 145 с. https://dspace.onua.edu.ua/items/1649eee6-5160-4a06-81c7-5614854f2390

ДОДАТОК А

Приклад коду backend частини

UserController.js:

```
import jwt from 'jsonwebtoken';
import bcrypt from 'bcrypt';
import { validationResult } from "express-validator";
import User from '../models/user.js';

export const register = async (req,res) => {
    try {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.status(400).json(errors.array());
        }

        const password = req.body.password;
        const salt = await bcrypt.genSalt(10);
        const hash = await bcrypt.hash(password, salt);

        const doc = new User({
            email: req.body.email,
            fullName: req.body.fullName,
            passwordHash: hash,
        });

        const user = await doc.save();

        const token = jwt.sign({
                _id: user._id,
            },
            'secret123,',
            {
                expiresIn: '30d'
            },
        );

        const {passwordHash, ...userData} = user._doc;

        res.json({
            ...userData,
            token,
        });
    } catch (error) {
        console.log(error);
        res.status(500).json({
            message: 'Failed to register',
        });
    }
};
```

```
export const login = async (req, res) => {
    try {
        const user = await User.findOne({email: req.body.email});

        if (!user) {
            return res.status(404).json({
                message: 'User not found',
            });
        }

        const isValidPass = await bcrypt.compare(req.body.password,
user._doc.passwordHash);

        if (!isValidPass) {
            return res.status(400).json({
                message: 'Wrong login or password',
            });
        }

        const token = jwt.sign({
                _id: user._id,
            },
            'secret123',
            {
                expiresIn: '30d'
            },
        );

        const {passwordHash, ...userData} = user._doc;

        res.json({
            ...userData,
            token,
        });
    } catch (error) {
        console.log(error);
        res.status(500).json({
            message: 'Failed to login',
        });
    }
};
```

CarsCatalog.js:

```
import Car from '../models/cars.js';

export const create = async (req, res) => {
    try {
        const doc = new Car({
            year: req.body.year,
            make: req.body.make,
            model: req.body.model,
```

```
            type: req.body.type,
            img: req.body.img,
            description: req.body.description,
            fuelConsumption: req.body.fuelConsumption,
            engineSize: req.body.engineSize,
            accessories: req.body.accessories,
            functionalities: req.body.functionalities,
            rentalPrice: req.body.rentalPrice,
            rentalCompany: req.body.rentalCompany,
            address: req.body.address,
            rentalConditions: req.body.rentalConditions,
            mileage: req.body.mileage
        });

        const post = await doc.save();

        res.json(post);
    } catch (error) {
        console.log(error);
        res.status(500).json({
            message: 'Failed to post car'
        });
    }
};

export const getAll = async (req, res) => {
    try {
        const { page = 1, itemsPerPage = 8, brand, priceFrom,
priceTo, mileageFrom, mileageTo } = req.query;
        const skip = (page - 1) * itemsPerPage;

        let filter = {};
        if (brand) {
            filter.make = brand;
        }
        if (priceFrom || priceTo) {
            filter.rentalPrice = {};
            if (priceFrom) {
                filter.rentalPrice.$gte = priceFrom;
            }
            if (priceTo) {
                filter.rentalPrice.$lte = priceTo;
            }
        }
        if (mileageFrom || mileageTo) {
            filter.mileage = {};
            if (mileageFrom) {
                filter.mileage.$gte = mileageFrom;
            }
            if (mileageTo) {
                filter.mileage.$lte = mileageTo;
            }
        }
```

```
        const catalog = await
Car.find(filter).skip(skip).limit(itemsPerPage);

        res.json(catalog);
    } catch (error) {
        console.log(error);
        res.status(500).json({
            message: 'Failed to get cars catalog'
        });
    }
};
```

ДОДАТОК Б

Приклад коду frontend частини

Catalog.jsx:

```
import React, { useEffect, useState } from "react";
import { useDispatch, useSelector } from 'react-redux';
import PropTypes from "prop-types";
import Gallery from "../../components/CarList";
import Loader from "../../components/Loader";
import CarSearch from "../../components/CarSearch";
import { Container, Button, PageForm, PageInfo } from
"./Catalog.styled";
import { fetchCatalog, setHasMore, setCurrentPage,
decreaseCurrentPage } from '../../redux/slices/catalog';
import { setFilter, resetFilter } from '../../redux/slices/filter';
import { setFilteredCars } from '../../redux/slices/catalog';

const Catalog = ({ setCars, favoriteToggle }) => {
  const dispatch = useDispatch();
  const { catalog } = useSelector(state => state.catalog);
  const filter = useSelector(state => state.filter);
  const [error, setError] = useState(null);
  const [isLoading, setIsLoading] = useState(false);
  const filteredCars = useSelector(state =>
state.catalog.filteredCars);
  const [pageNumber, setPageNumber] =
useState(catalog.currentPage.toString());

  useEffect(() => {
    const fetchData = async () => {
        try {
            setIsLoading(true);
            const favoriteCars = localStorage.getItem("favs")
                ? JSON.parse(localStorage.getItem("favs")).map((fav)
=> fav._id)
                : [];
            const data = await dispatch(fetchCatalog({
                page: catalog.currentPage,
                itemsPerPage: catalog.itemsPerPage,
                filter, // Передаем фильтр в запрос
            }));
            const favoritedCars = data.map((car) => ({
                ...car,
                favorite: favoriteCars.includes(car._id) ? true :
false,
            }));
            setCars(favoritedCars);
            dispatch(setFilteredCars(favoritedCars));
            if (data.length < catalog.itemsPerPage) {
                dispatch(setHasMore(false));
```

```
            } else {
                dispatch(setHasMore(true));
            }
        } catch (error) {
            setError(error.message);
        } finally {
            setIsLoading(false);
        }
    };

    fetchData();
}, [catalog.currentPage, catalog.itemsPerPage, setCars, dispatch,
filter]);

  useEffect(() => {
    if (filter.brand || filter.priceFrom || filter.priceTo ||
filter.mileageFrom || filter.mileageTo) {
      dispatch(setCurrentPage(1));
    }
  }, [filter, dispatch]);

  const loadMore = () => {
      dispatch(setCurrentPage(catalog.currentPage + 1));
  };

  const goBack = () => {
      dispatch(decreaseCurrentPage());
      dispatch(setHasMore(true));
  };

  const searchCars = (searchData) => {
    dispatch(setFilter(searchData));
    dispatch(setCurrentPage(1));
  };

  const resetSearch = () => {
    dispatch(resetFilter());
    dispatch(setCurrentPage(1));
    dispatch(setHasMore(true));
  };

  const handlePageChange = (e) => {
    setPageNumber(e.target.value);
  };

  const goToPage = (e) => {
    e.preventDefault();
    dispatch(setCurrentPage(Number(pageNumber)));
  };

  return (
      <Container>
          <CarSearch onSearch={searchCars} onReset={resetSearch} />
```

```
          {isLoading && <Loader />}
          {error && <div>{error.message}</div>}
          <Gallery cars={filteredCars} setFavorite={favoriteToggle}
/>
          <div>
            {catalog.currentPage > 1 && !isLoading && <Button
onClick={goBack}>Go Back</Button>}
            {catalog.hasMore && !isLoading && <Button
onClick={loadMore}>Load more</Button>}
          </div>
          <PageInfo>
            Page {catalog.currentPage}
          </PageInfo>
          <PageForm onSubmit={goToPage}>
            <input
              type="number"
              min="1"
              value={pageNumber}
              onChange={handlePageChange}
            />
            <button type="submit">Go</button>
          </PageForm>
      </Container>
  );
};

Catalog.propTypes = {
    setCars: PropTypes.func.isRequired,
    favoriteToggle: PropTypes.func.isRequired,
};

export default Catalog;
```

App.jsx:

```
import { Route, Routes, Navigate } from "react-router-dom";
import { useState } from "react";
import { lazy, Suspense } from "react";
import './Style.css';
import Loader from "./Loader";

// Импортируем страницы Registration и LogIn
const Registration = lazy(() => import("../pages/Registration"));
const LogIn = lazy(() => import("../pages/LogIn"));
const Home = lazy(() => import("../pages/Home"));
const Sidebar = lazy(() => import("./Header"));
const Catalog = lazy(() => import("../pages/Catalog"));
const Favorites = lazy(() => import("../pages/Favorites"));

export const App = () => {
  const [cars, setCars] = useState([]);
```

```
  const favoriteToggle = (_id) => {
    console.log("Toggled car with id:", _id);

    const updatedCars = cars.map((car) => ({
      ...car,
      favorite: car._id === _id ? !car.favorite : car.favorite,
    }));
    setCars(updatedCars);

    const favoriteCars = updatedCars.filter((car) => car.favorite
=== true);
    localStorage.setItem("favs", JSON.stringify(favoriteCars));
  };

  return (
    <Suspense fallback={<Loader />}>
      <Routes>
        <Route path="/" element={<Sidebar />}>
          <Route index element={<Home />} />
          <Route path="/registration" element={<Registration />} />
          <Route path="/login" element={<LogIn />} />
          <Route
            path="/catalog"
            element={
              <Catalog
                cars={cars}
                setCars={setCars}
                favoriteToggle={favoriteToggle}
              />
            }
          />
          <Route
            path="/favorites"
            element={<Favorites cars={cars}
favoriteToggle={favoriteToggle} />}
          />
        </Route>
        <Route path="*" element={<Navigate to="/" />} />
      </Routes>
    </Suspense>
  );
};
```

Додаток В

Копії документів про апробацію результатів роботи

СТВОРЕННЯ ВЕБ-ВОДАТКІВ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ MERN

Стоянов Артем Ігорович

студент 4-го курсу факультету кібербезпеки та інформаційних технологій

Національного університету «Одеська юридична академія»

У сучасному світі, веб-технології з кожним роком розвиваються дуже стрімкого, тому з'являються нові методи розробки веб-застосунків, що дають змогу створювати потужні продукти швидко та більш легко. Одним з таких методів є стек MERN.

Стек Mern – потужний інструмент у руках досвідчених розробників для створення та розгортання унікальних SPA застосунків. Він поєднує кілька технологій – MongoDb, Express.js, React.js, Node.js. З перших букв назв технологій складається абревіатура MERN [1].

Цей стек дозволяє створювати якісні, високофункціональні застосунки, враховуючи потреби замовників та досягнути максимальну зручність для взаємодії користувача із застосунком.

Давайте більш детально розглянемо технології, що використовуються у стеку MERN:

MongoDB – це база даних документів, розроблена для полегшення розробки та масштабування програм. Вона була обрана для керування базами даних [2].

Основні можливості MongoDB включають:

- Збереження даних у форматі JSON-подібних документів, що дозволяє гнучко організувати структуру даних;
- Використання гнучкої мови для формування запитів;
- Підтримка індексів для швидкого доступу до інформації;
- Функціонал профілювання запитів, для підвищення продуктивності;
- Ефективне зберігання різноманітних даних, таких як фотографії та відео;

– Ведення журналу операцій для моделювання баз даних.

Використання MongoDB дозволить легко вносити зміни у дані та їх структуру, не шкодячи вашому проекту, завдяки гнучкісті роботи з даними.

Express.js – це мінімалістичний та гнучкий фреймворк для веб-застосунків, побудованих на Node.js, що надає широкий набір функціональності. Маючи в своєму розпорядженні безліч допоміжних HTTP-методів та проміжних обробників, створювати надійні API можна легко і швидко [3].

Крім того, Express.js пропонує широкий набір додаткових модулів, які допомагають у реалізації інших функцій, таких як маршрутизація, обробка запитів форми та автентифікація.

React – це бібліотека JavaScript для відтворення інтерфейсів користувача. Інтерфейс користувача складається з невеликих елементів, таких як кнопки, текст і зображення. React дозволяє вам об'єднувати їх у багаторазово використовувані вкладені компоненти. Від веб-сайтів до програм для телефону, все на екрані можна розбити на компоненти [3]. Також він має такі переваги [4]:

– Швидкість програми збільшується завдяки внесенню мінімальних змін до реальної DOM завдяки використанню віртуальної DOM і ефективного методу оновлення;

– Дизайн компонентів, який лежить в основі JS React, дозволяє розділити інтерфейс на окремі компоненти. Це полегшує написання, тестування та підтримку коду, останній з яких можна легко змінювати та повторно використовувати;

– React заохочує односторонній потік даних, що робить керування станом програми легшим і передбачуваним, а також полегшує тестування та налагодження програм;

– Існує велика та активна спільнота розробників, які використовують React. Багато інструментів розробки, бібліотек і ресурсів. Крім того, велика кількість сторонніх бібліотек допомагає React і розширює його функції.

Node.js – це безкоштовне міжплатформне середовище виконання JavaScript із відкритим кодом, яке дозволяє розробникам створювати сервери, веб-

програми, інструменти командного рядка та сценарії. Node.js має унікальну перевагу, оскільки мільйони інтерфейсних розробників, які пишуть JavaScript для браузера, тепер можуть писати код на стороні сервера на додаток до коду на стороні клієнта без необхідності вивчати абсолютно іншу мову [5].

Завдяки широкому спектру модулів і пакетів, доступних через npm, Node.js дозволяє розробникам швидко створювати широкий спектр програм, включаючи програми баз даних, веб-сервери до інструментів командного рядка. Node.js використовується багатьма великими підприємствами, через свою, через свою ефективність, швидкодію та простоту використання.

MERN займає провідне місце в галузі розробки додатків завдяки таким факторам:

– Кожен рядок коду пишеться на JavaScript, що позбавляє необхідності перемикати контекст.;

– Стек MERN сприяє зростанню ідеальних умов для створення високопродуктивних web-додатків;

– MERN пропонує широкий спектр можливостей тестування, які охоплюють весь життєвий цикл розробки програмного забезпечення.

Технологічний стек MERN є дуже корисним інструментом для розробки сучасних веб-додатків. Він поєднує різноманітні інструменти та технології, що дозволяє розробникам створювати динамічні та ефективні онлайн-додатки з єдиною мовою програмування для використання як на стороні клієнта, так і на стороні сервера. Будь то великий бізнес-проект чи маленька персональна веб-сторінка, у MERN є все необхідне для належної реалізації вашої ідеї.

Список використаних джерел:

1. Створення сайтів та застосунків на MERN: http://surl.li/tteih
2. MongoDB – Official website: https://www.mongodb.com/docs/manual/
3. Express.js – Official website: https://expressjs.com/uk/
4. React.js – Official Website: https://react.dev/learn/describing-the-ui
5. Node.js – Official website: https://nodejs.org/en/

Ключові слова: MERN, стек, сервер, веб

Keywords: MERN, stack, server, web

Науковий керівник: доцент Лобода Ю.Г.