МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Відокремлений структурний підрозділ
«Тернопільський фаховий коледж
Тернопільського національного технічного університету імені Івана Пулюя»

Відділення телекомунікацій та електронних систем

Циклова комісія комп'ютерних наук

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра

на тему: Розробка вебзастосунку для автоматизації управління автопарком «Burunduk Garage»

Виконав: студент IV курсу, групи КН-421
спеціальності: 122 Комп'ютерні науки
Максим БРИДУН

Керівник Галина РАДЧИК

Рецензент
(ім'я та прізвище)

м. Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
ІМЕНІ ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем
Циклова комісія комп'ютерних наук
Освітньо-професійний ступінь «фаховий молодший бакалавр»
Спеціальність 122 Комп'ютерні науки
Галузь знань 12 Інформаційні технології

ЗАТВЕРДЖУЮ
Голова циклової комісії комп'ютерних наук
_____ Галина МАРЦІЯШ
« 02 » березня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бридуну Максиму Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка вебсайту для автоматизації управління автопарком «Burunduk Garage»

керівник роботи Радчик Галина Іванівна,
(прізвище, ім'я, по батькові)
затверджені наказом вищого навчального закладу № 4/9-132 від 27.02.2026 р.
2. Строк подання студентом роботи: 19.06.2026 р.
3. Вихідні дані до роботи: технічне завдання на розробку програмного забезпечення, мова програмування: JavaScript; фреймворки: Express.js; бібліотека React, Vite, стандарти IEEE 29148-2018, IEEE 29119, ДСТУ 8302:2015.

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

1 Загальний розділ
1.1 Аналітичний огляд існуючих рішень
1.2 Технічне завдання
1.2.1 Найменування та область застосування
1.2.2 Призначення розробки
1.2.3 Вимоги до функціоналу вебзастосунку
1.2.4 Вимоги до програмної документації
1.2.5 Техніко-економічні показники
1.2.6 Стадії та етапи розробки
1.2.7 Порядок тестування та прийому

2 Розробка технічного та робочого проєкту
2.1 Розробка структури сайту і web-сторінок
2.2 Створення та верстка сторінок сайту
2.3 Розробка структури бази даних сайту
2.4 Програмування сайту
2.4.1 Написання клієнтської частини
2.4.2 Написання admin-частини
2.5 Тестування web-сайту
3 Спеціальний розділ
3.1 Інструкція з розміщення сайту в Інтернеті
3.2 Інструкція з обслуговування та наповнення сайту
3.3 Інструкція з популяризації та підтримки сайту
4 Економічний розділ
4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР
4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи
4.3 Розрахунок витрат на електроенергію
4.4 Розрахунок суми амортизаційних відрахувань вебзастосунку «Burunduk Garage»
4.5 Обчислення накладних витрат
4.6 Складання кошторису витрат та визначення собівартості вебзастосунку «Burunduk Garage»
4.7 Розрахунок ціни вебзастосунку «Burunduk Garage»
4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень
5 Охорона праці, техніка безпеки та екологічні вимоги
5.1 Несприятливі виробничі фактори
5.2 Захисні засоби з електробезпеки
6 Висновки

Додаткові вказівки: виконання кваліфікаційної роботи із розробкою програмного продукту – вебсайту.

5. Перелік графічного матеріалу:

1. UML-діаграма варіантів використання web-застосунку
2. ER-діаграма бази даних web-застосунку
3. Структурна схема web-застосунку
4. Таблиця техніко-економічних показників

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Любов КАЛУШКА		
Охорона праці, техніка безпеки та екологічні вимоги	Генадій ГОРЯЧЕК		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	20.03.2026	
2	Збір і узагальнення інформації	01.05.2026	
3	Написання першого розділу	15.05.2026	
4	Розробка технічного та робочого проєкту	29.05.2026	
5	Написання спеціального розділу	05.06.2026	
6	Розрахунок економічної частини	08.06.2026	
7	Написання розділу охорони праці	09.06.2026	
8	Виконання графічної частини	10.06.2026	
9	Оформлення пояснювальної записки	11.06.2026	
10	Погодження нормоконтролю	12.06.2026	
11	Попередній захист кваліфікаційної роботи	11.06.2026	
12	Захист кваліфікаційної роботи	23.06.2026	

7. Дата видачі завдання: 05 березня 2026 р.

Студент _____ Максим БРИДУН
(підпис)

Керівник роботи _____ Галина РАДЧИК
(підпис)

ЗМІСТ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Розробка вебзастосунку для автоматизації управління автопарком «Burunduk Garage».

Метою кваліфікаційної роботи є розробка web-застосунку для автоматизації управління автопарком «Burunduk Garage», що забезпечує ведення каталогу автомобілів, оформлення та редагування бронювань, облік користувачів, роботу адміністративної панелі, перегляд статистики та зручну взаємодію клієнта із сервісом оренди автомобілів.

Пояснювальна записка складається з п’яти розділів.

У загальній частині описуються аналітичний огляд існуючих рішень у сфері автоматизації автопрокату та аналіз технічного завдання на розробку web-сайту «Burunduk Garage».

У другому розділі представлено процес створення програмного продукту, опис та обґрунтування вибору структури сайту і web-сторінок, створення та верстку сторінок, розробку структури бази даних сайту, програмування клієнтської та admin-частини, а також тестування web-сайту.

В спеціальній частині описані інструкція з розміщення сайту в Інтернеті, інструкція з обслуговування та наповнення сайту, а також інструкція з популяризації та підтримки web-сайту після впровадження.

Розрахунок вартості розробки та економічної ефективності приведено в економічній частині.

Основні питання охорони праці, техніки безпеки та екологічні вимоги розглянуто в п’ятому розділі.

Обсяг пояснювальної записки 98 сторінок.

До складу кваліфікаційної роботи входить графічна частина, яка складається зі структурної схеми web-сайту, схеми бази даних, блок-схеми алгоритму бронювання автомобіля, схеми взаємодії клієнтської частини, серверної частини, MongoDB та ImageKit, техніко-економічних показників і лістингів основних програмних модулів, що виконані на окремих аркушах формату А1.

ANNOTATION

Qualification work topic: Development of a web application for automating fleet management of «Burunduk Garage».

The purpose of the qualification work is to develop a web application for automating fleet management of «Burunduk Garage», which provides vehicle catalog management, creation and modification of reservations, user management, administrative panel functionality, statistical data analysis, and convenient customer interaction with the car rental service.

The explanatory note consists of five sections.

The first section contains an analytical review of existing solutions in the field of car rental automation and an analysis of the technical requirements for the development of the «Burunduk Garage» website.

The second section presents the software development process, describes and justifies the selected website structure and web pages, page creation and layout, database design, development of the client-side and administrative modules, as well as website testing.

The special section includes instructions for website deployment on the Internet, guidelines for website maintenance and content management, and recommendations for website promotion and support after implementation.

The calculation of development costs and economic efficiency is presented in the economic section.

The main issues of occupational safety, labor protection, and environmental requirements are considered in the fifth section.

The explanatory note contains 98 pages.

The qualification work also includes a graphical part consisting of a website structural diagram, a database schema, a vehicle booking algorithm flowchart, an interaction diagram of the client side, server side, MongoDB, and ImageKit services, technical and economic indicators, and listings of the main software modules presented on separate A1-format sheets.

ВСТУП

Сфера оренди автомобілів упродовж останніх років поступово відходить від ручного обліку, телефонних домовленостей і розрізнених електронних таблиць. Клієнт очікує швидко побачити автопарк, порівняти автомобілі й самостійно вибрати дати, а працівник сервісу має тримати під контролем бронювання, статуси та зміни в даних без постійного ручного перенесення інформації.

Актуальність теми зумовлена тим, що навіть невеликий автопарк потребує узгодженої інформаційної системи. Коли дані про автомобілі, клієнтів, оренди, ціни та статуси зберігаються у різних джерелах, адміністратор витрачає багато часу на перевірку доступності машин, уточнення дат, оновлення фотографій і обробку змін у бронюваннях. Такі процеси особливо вразливі до людського фактора: одна помилка в датах або статусі може призвести до подвійного бронювання чи незручності для клієнта.

Вебзастосунок «Burunduk Garage» розробляється як інструмент, що об'єднує публічну частину сервісу та адміністративну панель. Публічна частина потрібна для перегляду автопарку, ознайомлення з характеристиками автомобілів, вибору дат і створення бронювання. Адміністративна частина потрібна для додавання та редагування автомобілів, керування бронюваннями, пошуку клієнтів, фільтрації за датами, контролю статусів і перегляду статистики доходу.

Особливістю проєкту є те, що він не обмежується статичним каталогом автомобілів. У системі реалізовано логіку оренди, перевірку доступності автомобіля на вибраний проміжок, редагування додаткових опцій, зміну дат оренди, скасування бронювання за визначеним правилом, а також різні сценарії для клієнтської частини та службового кабінету. Це наближає застосунок до реального робочого процесу сервісу прокату автомобілів.

Метою кваліфікаційної роботи є розробка вебзастосунку для автоматизації управління автопарком «Burunduk Garage», який забезпечує зручну взаємодію клієнта з сервісом оренди та надає адміністратору засоби для керування основними операційними процесами. Досягнення цієї мети передбачає не лише створення

інтерфейсу, а й проєктування структури даних, API, ролей доступу, перевірок бізнес-правил і сценаріїв тестування.

Щоб досягти поставленої мети, потрібно виконати такі завдання: проаналізувати предметну область та існуючі підходи до автоматизації автопрокату; сформувати технічне завдання; визначити функціональні й нефункціональні вимоги; спроєктувати структуру бази даних; розробити серверну частину для роботи з автомобілями, користувачами, бронюваннями і статистикою; створити клієнтський інтерфейс для користувача та адміністратора; реалізувати адаптивність для різних пристроїв; виконати тестування основних сценаріїв.

Об'єктом дослідження є процеси управління автопарком і оформлення оренди автомобілів. Предметом дослідження є методи та програмні засоби створення клієнт-серверного вебзастосунку, який автоматизує ці процеси. У межах роботи розглядаються питання структурування даних, побудови API, захисту доступу, реалізації інтерфейсу, обробки зображень і забезпечення зрозумілого користувацького досвіду.

Під час розробки використано сучасний стек вебтехнологій. React застосовано для побудови компонентного інтерфейсу, Vite - для швидкої розробки та збірки, Tailwind CSS - для стилізації, Express - для серверної логіки, MongoDB і Mongoose - для зберігання та моделювання даних, JWT - для авторизації, ImageKit - для обробки та зберігання фотографій автомобілів. Такий стек дозволяє побудувати гнучку систему, яку можна розширювати після завершення кваліфікаційної роботи.

Практична цінність роботи полягає у створенні програмного продукту, що демонструє повний цикл роботи сервісу оренди автомобілів: від публікації транспортного засобу в автопарку до створення бронювання, його редагування, скасування та відображення в адміністративній панелі. Система може бути використана як основа для реального сервісу, а також як навчальний приклад розробки повноцінного вебзастосунку з клієнтською, серверною та адміністративною частинами.

У першому розділі пояснювальної записки подано аналітичний огляд існуючих рішень і сформовано технічне завдання. У другому розділі описано структуру сайту, базу даних, алгоритми, інформаційні зв'язки, програмну реалізацію та тестування. У третьому розділі розглянуто розміщення вебзастосунку, інструкції з адміністрування, наповнення, підтримки і популяризації. У висновках підсумовано результати виконаної роботи.

Під час роботи над системою довелося орієнтуватися на щоденні дрібні дії, які найчастіше виконуються в автопрокаті: знайти бронювання за клієнтом, перевірити найближчі дати, змінити статус, оновити автомобіль або підправити фото в картці. З боку клієнта основним залишився короткий шлях: переглянути авто, вибрати період, оформити бронювання і за потреби змінити його в дозволений термін.

Розробка вебзастосунку «Burunduk Garage» є актуальною саме через практичність задачі: автопарк потребує не просто гарної сторінки з автомобілями, а робочого інструмента для бронювань, клієнтів, дат, фото та статистики. Реалізована система зменшує кількість ручних дій і створює основу для подальшого розвитку сервісу.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Автоматизація сервісу оренди автомобілів може виконуватися різними способами. Найпростішим варіантом є використання електронних таблиць, календарів і месенджерів. Цей варіант не потребує складної розробки, однак він не забезпечує єдиного джерела даних, не має автоматичної перевірки перетину дат і не дозволяє клієнту самостійно керувати бронюванням.

Другим підходом є використання універсальних CRM-систем. Вони дають змогу вести клієнтів, нотатки, статуси й іноді інтегрувати календарі, проте не завжди враховують специфіку автопрокату: характеристики автомобілів, сезонність, різні типи пального, додаткові опції, логіку доступності за датами та взаємозв'язок між автомобілем і конкретним періодом оренди.

Третій підхід - використання готових платформ для бронювання транспорту. Такі рішення можуть мати значну кількість функцій, проте часто потребують абонентської оплати, обмежують дизайн, ускладнюють локалізацію під конкретний бренд і не завжди дозволяють змінювати бізнес-логіку. Для невеликого бренду це може бути надлишковим або економічно недоцільним.

Четвертий підхід - розробка власного вебзастосунку. Він потребує більше часу на проєктування та реалізацію, однак дає змогу точно врахувати потреби автопарку, адаптувати інтерфейс під бренд, закласти необхідні правила бронювання, створити адміністративну панель і поступово розширювати систему відповідно до реальних процесів.

Для «Burunduk Garage» доцільним є саме індивідуальний вебзастосунок, оскільки він поєднує каталог автомобілів, особистий кабінет користувача, адміністративний облік бронювань, статистику та роботу із зображеннями. У готових сервісах подібні дрібні налаштування часто довелося б обходити сторонніми рішеннями, а тут їх можна закласти одразу в логіку проєкту.

Сервісу оренди автомобілів недостатньо просто показати каталог машин:

потрібно пов'язати наявність автомобіля, дати оренди, клієнтські дані та дії працівника. Багато невеликих автопарків починають роботу з таблиць або месенджерів, але зі збільшенням кількості клієнтів такий облік стає незручним.

Під час аналізу предметної області було визначено, що основна складність автопрокату полягає у роботі з часовими проміжками. Один автомобіль не може бути заброньований двома клієнтами на перетин дат, тому система повинна перевіряти доступність не тільки за статусом автомобіля, а й за фактичними періодами оренди, які вже збережені в базі даних.

Готові CRM-системи дозволяють вести клієнтів і нотатки, але вони не завжди містять спеціалізовані поля для характеристик автомобілів, типів пального, витрати, класів, фотографій, опцій оренди та фільтрації за датами. Для Burunduk Garage важливо було отримати саме спеціалізований інтерфейс, у якому адміністратор працює не з абстрактними заявками, а з реальним автопарком.

Готові платформи бронювання транспорту можуть бути функціональними, однак вони часто мають обмеження у зміні дизайну, структурі карток автомобілів і правилах редагування оренди. У цьому проєкті потрібно було врахувати власну логіку: редагування та скасування для користувача тільки за два дні до початку оренди, окремий пошук у бронюваннях і гнучкі підписи характеристик без зміни серверної моделі.

Порівняно з універсальними рішеннями власний web-застосунок дає змогу адаптувати інтерфейс під конкретний бренд. Для Burunduk Garage це проявляється у картках автомобілів, матовому фоні, налаштуванні положення фотографії, автоматичній трансформації зображень через ImageKit і поділі сторінок на клієнтську та адміністративну частини.

Окремим фактором є зручність адміністратора. У реальній роботі адміністратор часто шукає бронювання за прізвищем клієнта, назвою автомобіля або періодом оренди. Тому сторінка бронювань повинна підтримувати пошук без перезавантаження сторінки, фільтр за датами, фільтр за статусом, а також швидкий перегляд майбутніх і завершених оренд.

Для клієнта найважливішими є швидкість вибору автомобіля, зрозумілі характеристики, коректні дати та можливість переглянути свої оренди після оформлення. Якщо система дозволяє самостійно змінити додаткові опції або дату в допустимий період, зменшується кількість ручних звернень до адміністратора і підвищується довіра до сервісу.

Отже, аналіз існуючих рішень показав доцільність розробки власного web-застосунку. У власному рішенні можна поєднати каталог автопарку, систему бронювання, адміністративну панель, статистику, обробку фотографій та адаптивний інтерфейс в одному програмному продукті, який відповідає задачам Burunduk Garage.

Щоб обґрунтувати вибір власної розробки, різні підходи до автоматизації автопрокату потрібно порівняти за перевагами та обмеженнями. Таке порівняння показує, чому для Burunduk Garage доцільним є окремий web-застосунок, а не використання електронних таблиць або універсальної CRM-системи. Результати порівняння наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння підходів до автоматизації автопрокату

Підхід	**Переваги**	**Недоліки**
Електронні таблиці	Швидкий старт, мінімальні витрати	Високий ризик помилок, відсутність автоматичної перевірки дат, немає клієнтського інтерфейсу
CRM-система	Облік клієнтів, статуси, нотатки	Не враховує всі особливості автопрокату, потребує налаштування
Готова платформа	Багато вбудованих функцій	Залежність від сервісу, оплата, обмежений контроль дизайну і логіки
Власний вебзастосунок	Повна відповідність задачам, контроль даних і дизайну	Потребує проєктування, розробки та підтримки

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Повне найменування програмного продукту: вебзастосунок для автоматизації управління автопарком «Burunduk Garage». Скорочене найменування: Burunduk Garage.

Область застосування - організація роботи сервісу оренди автомобілів. Система може використовуватися у невеликому або середньому автопарку, де необхідно вести каталог автомобілів, приймати бронювання, зберігати дані користувачів, контролювати статуси оренд і переглядати фінансову статистику.

Назва Burunduk Garage використовується як умовне найменування сервісу, навколо якого побудовано інтерфейс і логіку web-сайту. Система орієнтована на автопарк, де адміністратор повинен швидко змінювати інформацію про автомобілі, а клієнт має отримувати актуальні дані без додаткових дзвінків чи повідомлень.

Область застосування web-застосунку охоплює операції, які повторюються щоденно: додавання нових автомобілів, оновлення фотографій, перевірка доступності, створення бронювань, зміна статусів, редагування дат, перегляд статистики та робота з користувачами. Саме ці процеси були покладені в основу функціональної структури проєкту.

1.2.2 Призначення розробки

Експлуатаційне призначення розробки полягає у наданні користувачам вебінтерфейсу для перегляду автомобілів, вибору дат оренди, оформлення бронювання і керування власними орендами. Система повинна бути зрозумілою для користувача без додаткового навчання.

Функціональне призначення полягає в автоматизації рутинних операцій: розрахунку вартості оренди, перевірки перетину дат, збереження додаткових опцій, редагування бронювань із часовим обмеженням, пошуку та фільтрації в адміністративних таблицях.

Розробка призначена для зменшення ручної роботи адміністратора та створення єдиного інформаційного середовища для автопарку. Замість окремих таблиць, чатів і записів система зберігає автомобілі, клієнтів і бронювання в базі даних, а всі основні дії виконуються через web-інтерфейс.

Для клієнта призначення системи полягає у простому виборі автомобіля та контролі власних оренд. Користувач може переглянути характеристики, вибрати період, оформити бронювання, а після цього побачити його у панелі «Мої оренди». Такий сценарій робить послугу оренди більш прозорою.

У службовому кабінеті система використовується для керування даними автопарку. Тут редагуються автомобілі, переглядаються бронювання, працюють пошук і фільтрація, відкривається список користувачів та статистика. Завдяки цьому зміни можна вносити швидше, не розкидаючи інформацію між таблицями і повідомленнями.

1.2.3 Вимоги до функціоналу web-сайту

Web-сайт повинен забезпечувати повний цикл взаємодії клієнта із сервісом оренди автомобілів: перегляд автопарку, фільтрацію та сортування автомобілів, відкриття детальної сторінки, вибір дат, перевірку доступності, оформлення бронювання і перегляд власних оренд.

Для звичайного користувача необхідно реалізувати реєстрацію, вхід, особисту панель «Мої оренди», редагування дат та додаткових опцій, а також можливість скасування бронювання тільки за два дні до початку оренди. Таке обмеження потрібне для стабільного планування роботи автопарку.

Адміністративна частина web-сайту повинна містити захищені сторінки керування автомобілями, користувачами, бронюваннями та статистикою. На сторінці бронювань мають працювати пошук за автомобілем або клієнтом, фільтрація за датами оренди, фільтрація за статусами, включно з майбутніми та завершеними бронюваннями.

Функціонал автопарку повинен підтримувати додавання, редагування та

видалення автомобілів, сортування за розходом палива, роботу з різними типами пального, обробку фотографій через ImageKit і налаштування положення зображення у картці за допомогою параметрів X, Y та масштабу.

Інтерфейс web-сайту має бути адаптивним для комп'ютерів і мобільних пристроїв, зрозумілим для користувача, стійким до помилок введення та достатньо плавним під час прокручування сторінок.

Функціонал web-сайту повинен бути розділений відповідно до ролей. Незареєстрований відвідувач може переглядати основну інформацію та сторінки автомобілів, зареєстрований користувач отримує можливість створювати бронювання, а адміністратор має доступ до службових сторінок, які не повинні відкриватися звичайним користувачам.

На сторінці автопарку обов'язковими є фільтрація та сортування. Система повинна давати змогу швидко знайти автомобіль за класом, типом кузова, типом пального, ціною або іншими характеристиками. Сортування за розходом палива має працювати як числове, щоб значення з дробовою частиною не опинялися після цілих чисел через рядкове порівняння.

На сторінці деталей автомобіля користувач повинен бачити фото, характеристики, ціну за день, доступні опції та форму вибору дат. Після вибору періоду система має перевірити доступність автомобіля й не дозволити створити бронювання, якщо обрані дати перетинаються з активною орендою.

У панелі «Мої оренди» користувач повинен бачити всі свої бронювання та їхній стан. Редагування дат, зміна додаткових опцій і скасування мають бути доступні тільки за два дні до початку оренди. Це правило реалізує баланс між зручністю клієнта та стабільністю роботи автопарку.

Admin-частина повинна містити сторінку бронювань із пошуком за клієнтом або автомобілем, фільтрами за датами оренди та статусами. У практичній роботі пошук має не перезавантажувати сторінку після введення кожної літери, а працював плавно через стан компонента або контрольований запит.

Функціонал роботи із зображеннями повинен підтримувати вставлення

посилання на фото, обробку через ImageKit і збереження URL, який уже містить трансформацію видалення фону. Додатково адміністратор має мати можливість змінювати положення фото по горизонталі, вертикалі та масштаб, щоб картка виглядала акуратно.

Окремо передбачено режим гостя. Відвідувач без авторизації може переглядати головну сторінку, автопарк і сторінки автомобілів, але не може оформити бронювання або відкрити особисту панель. Такий режим потрібний, щоб людина спочатку ознайомилась із сервісом, а реєстрацію проходила вже тоді, коли справді хоче орендувати автомобіль.

1.2.4 Вимоги до програмної документації

Програмна документація повинна містити пояснювальну записку, опис предметної області, технічне завдання, структуру web-сайту, опис бази даних, алгоритмів, програмної реалізації, тестування, інструкції з розміщення та обслуговування сайту.

У документації необхідно описати призначення основних модулів: клієнтської частини, admin-частини, серверної частини, моделей бази даних, API-маршрутів, авторизації, бронювання, статистики та інтеграції з ImageKit.

Графічна частина документації має містити місця для скриншотів сторінок, схеми структури web-сайту, схеми бази даних, блок-схеми алгоритмів і приклади інтерфейсів адміністратора та користувача.

У додатках доцільно подати лістинги основних файлів проєкту, які підтверджують практичну реалізацію: моделі бази даних, серверні маршрути, компоненти React, сервіс API та допоміжні утиліти.

Програмна документація повинна пояснювати не тільки кінцевий результат, а й послідовність його створення. У пояснювальній записці необхідно показати, чому було обрано клієнт-серверну архітектуру, які сутності зберігаються в базі даних і як компоненти взаємодіють між собою.

Опис клієнтської частини має містити призначення основних сторінок, логіку

маршрутизації, роботу з API, формування карток автомобілів, модальні вікна бронювання та сторінку «Мої оренди». Такий опис потрібний для розуміння того, як користувач взаємодіє з системою.

Опис admin-частини має містити сторінки автомобілів, користувачів, бронювань і статистики. У документації важливо окремо пояснити захист адміністративних маршрутів, оскільки доступ до них впливає на цілісність даних і безпеку роботи всього web-сайту.

У додатках до документації доцільно подати фрагменти коду моделей, серверних маршрутів, сервісу API, компонентів автопарку, форми автомобіля, сторінки бронювань і утиліт форматування характеристик. Це підтверджує, що описана функціональність реалізована у програмному коді.

1.2.5 Техніко-економічні показники

Техніко-економічні показники web-сайту визначають доцільність його розробки та подальшого використання в роботі автопарку. Основними показниками є трудомісткість розробки, орієнтовна собівартість, витрати на підтримку, очікувана економія часу адміністратора і можливість масштабування системи. Зведену таблицю техніко-економічних показників проєкту зображено у графічній частині 2026.КВР.122.421.02.00.00. ТБ.

Розробка web-сайту виконується з використанням відкритих технологій React, Vite, Express, MongoDB і Mongoose, що зменшує витрати на ліцензійне програмне забезпечення. Додаткові витрати можуть бути пов'язані з хостингом, доменом, зберіганням зображень в ImageKit і підтримкою бази даних.

Економічний ефект досягається завдяки зменшенню ручної роботи адміністратора, швидшому пошуку, автоматичному обліку дат, зменшенню помилок під час планування оренди та зручнішій взаємодії клієнта з автопарком.

Техніко-економічні показники проєкту визначаються насамперед трудомісткістю розробки та очікуваною користю від автоматизації. Для Burunduk Garage найбільший ефект полягає у скороченні часу на ручну перевірку бронювань,

пошук клієнтів, оновлення інформації про автомобілі та формування загальної статистики.

Використання React, Express і MongoDB зменшує витрати на ліцензійне програмне забезпечення, оскільки основні технології є відкритими. При цьому система залишається достатньо гнучкою: у майбутньому можна додати онлайн-оплату, календар зайнятості, повідомлення клієнтам або інтеграцію з іншими сервісами.

Економічна доцільність також пов'язана зі зменшенням помилок. Якщо адміністратор вручну записує дати оренди, існує ризик подвійного бронювання або неправильного статусу. Автоматична перевірка перетину дат і єдина база даних зменшують такі ризики та підвищують якість обслуговування.

1.2.6 Стадії та етапи розробки

Першою стадією була постановка задачі та уточнення функціональних вимог. На цьому етапі визначено, що web-сайт повинен мати три ролі користувачів, публічний автопарк, систему бронювання, admin-панель, сторінку статистики та можливість редагування фотографій автомобілів.

Другою стадією стало проєктування структури даних і сторінок. Було визначено моделі користувача, автомобіля, бронювання, відгуку та довідників. Одночасно сформовано структуру клієнтського інтерфейсу: автопарк, деталі автомобіля, форма бронювання, мої оренди, admin-сторінки.

Третьою стадією була реалізація та інтеграція. На цьому етапі створено серверні маршрути, підключено MongoDB, реалізовано авторизацію, компоненти React, роботу з ImageKit, фільтри, пошук, статистику.

Для контролю виконання роботи етапи розробки були згруповані за послідовністю виконання: від аналізу вимог до тестування та підготовки звітних матеріалів. Це дозволяє показати не лише перелік робіт, а й результат кожної стадії. Стадії та етапи розробки наведено у таблиці 1.2.

Таблиця 1.2 – Стадії та етапи розробки

Стадія	Зміст	Результат
Аналіз	Визначення задач автопрокату та ролей користувачів	Сформовані вимоги
Проєктування	Моделювання даних, API, маршрутизації та структури сторінок	Технічний проєкт
Реалізація	Створення компонентів, серверних маршрутів і моделей	Робочий вебзастосунок
Налагодження	Перевірка сценаріїв, виправлення помилок, адаптивність	Стабільна версія
Підготовка документації	Опис реалізації, тестування та інструкцій	Пояснювальна записка

1.2.7 Порядок тестування та прийому

Приймання програмного продукту виконується шляхом перевірки основних сценаріїв користувача й адміністратора. У користувацькому сценарії перевіряється реєстрація, вхід, перегляд автопарку, фільтрація, відкриття сторінки автомобіля, вибір дат, створення бронювання, редагування оренди та скасування за правилом двох днів.

В адміністративних сценаріях перевіряється доступ до захищених сторінок, додавання і редагування автомобіля, обробка зображення через ImageKit, зміна позиції фотографії, пошук користувачів, пошук і фільтрація бронювань, зміна статусів, перегляд статистики та коректність діаграми доходу по місяцях.

Приймання web-сайту доцільно виконувати за сценаріями, які повторюють реальну роботу користувачів. Спочатку перевіряється реєстрація, вхід, перегляд

автопарку, фільтрація, відкриття сторінки деталей, вибір дат і створення бронювання.

Окремо перевіряється логіка редагування бронювання. Якщо до початку оренди залишилося більше двох днів, користувач повинен мати можливість змінити дати або опції. Якщо часу менше, система повинна заборонити редагування та скасування.

В admin-частині перевіряється доступ тільки для ролі admin, коректність пошуку, фільтрації за датами, зміни статусів, редагування автомобіля, завантаження фото через ImageKit і відображення статистики. Після успішного проходження цих сценаріїв web-сайт можна вважати придатним для демонстрації.

Додатково під час приймання потрібно перевірити стабільність роботи сайту на різних пристроях. На комп'ютері оцінюється правильність відображення сторінок, таблиць, фільтрів і модальних вікон, а на мобільному пристрої перевіряється адаптивність автопарку, карток автомобілів, кнопок, форм авторизації та сторінки «Мої оренди». Це важливо, оскільки користувачі можуть оформлювати бронювання не лише з комп'ютера, а й безпосередньо з телефона.

Результатом приймання є підтвердження того, що web-сайт виконує основні функції відповідно до технічного завдання. Якщо під час перевірки виявляються помилки, вони фіксуються, після чого виконується їх виправлення і повторне тестування відповідного сценарію. Програмний продукт вважається прийнятим тоді, коли основні користувацькі та адміністративні дії виконуються без критичних помилок, а дані коректно зберігаються і відображаються в системі.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка структури сайту і web-сторінок

Структура вебзастосунку побудована навколо двох груп сторінок: публічних і адміністративних. До публічних належать головна сторінка, автопарк, сторінка деталей автомобіля, форма входу, форма реєстрації, сторінка підтримки та панель «Мої оренди». До адміністративних належать сторінки керування автомобілями, користувачами, бронюваннями і статистикою. Розподіл можливостей між гостем, зареєстрованим користувачем і адміністратором зображено у графічній частині 2026.КВР.122.421.02.00.00. ДВ.

Структурна схема web-сайту відображає головні розділи застосунку, логіку переходів головного меню та поділ функцій між головною сторінкою, автопарком, бронюваннями, адміністративною панеллю, авторизацією і підтримкою. Структурну схему сайту зображено у графічній частині 2026.КВР.122.421.02.00.00. СС.

Навігація організована так, щоб користувач швидко переходив від перегляду автопарку до вибору конкретного автомобіля. Адміністративні сторінки доступні лише користувачам з відповідною роллю. Це дозволяє відокремити клієнтський сценарій від службових операцій і зменшити ризик випадкового доступу до керування даними.

Карточка автомобіля містить фотографію, назву, клас, тип пального, ключові характеристики, ціну та елементи переходу до детального перегляду. Підписи характеристик зроблені достатньо гнучкими, щоб одна й та сама картка могла коректно показувати різні типи автомобілів.

Після визначення загальної структури сайту було підготовлено основні екрани, через які проходить користувач під час роботи з сервісом. Головна сторінка виконує роль першої точки входу: вона знайомить клієнта з брендом Burunduk Garage і спрямовує його до перегляду автопарку. Її вигляд показано на рисунку 2.1.

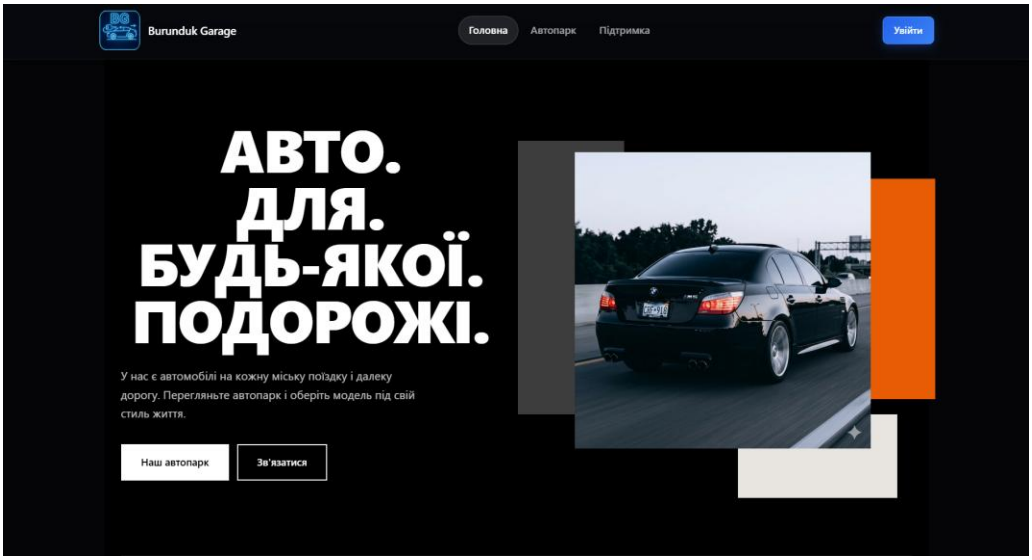


Рисунок 2.1 – Головна сторінка вебзастосунку

Наступним кроком користувач переходить до каталогу автомобілів, де може порівняти пропозиції за ціною, класом, типом пального та іншими характеристиками. Сторінку автопарку наведено на рисунку 2.2.

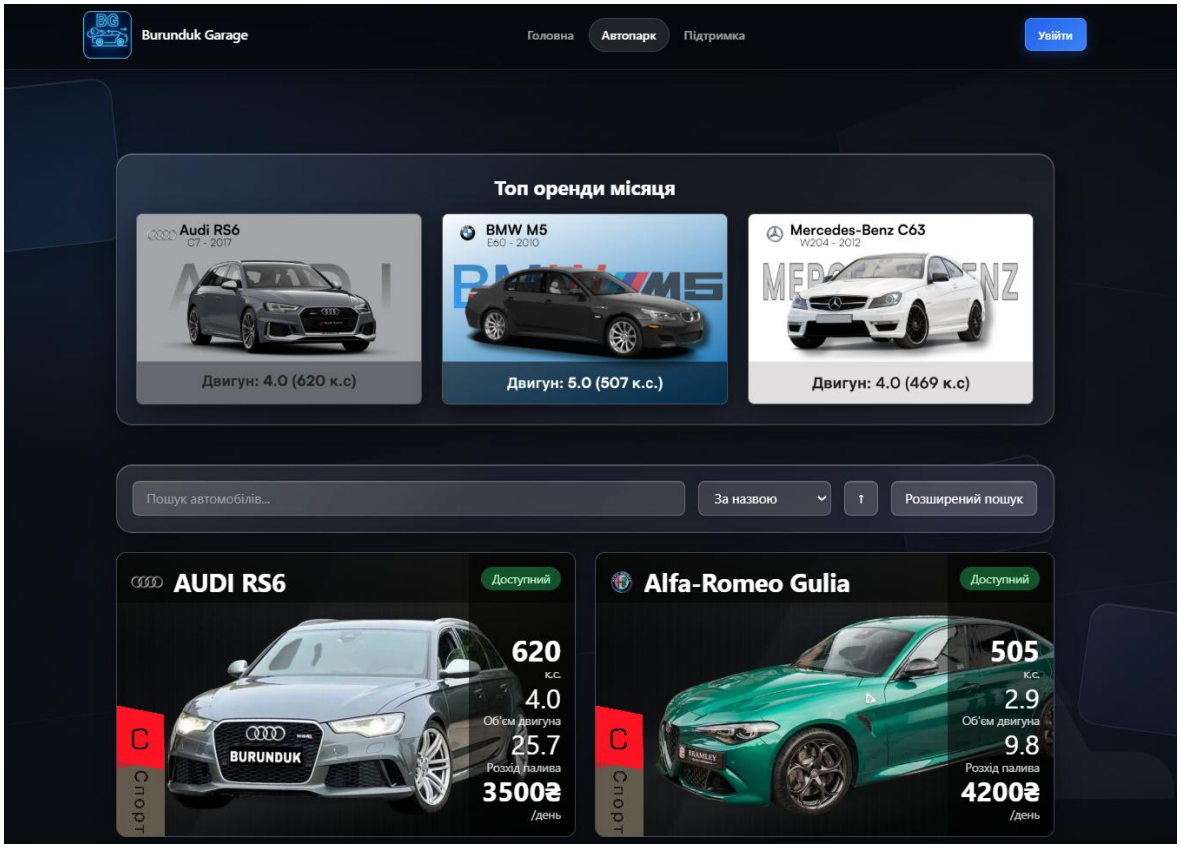


Рисунок 2.2 – Сторінка автопарку

Для вибраного автомобіля передбачено окрему сторінку з детальними характеристиками, фотографією та переходом до оформлення оренди. На цій сторінці користувач може переглянути основні параметри автомобіля, його вартість за добу, тип кузова, клас, характеристики двигуна або батареї, витрату та доступний статус. Також тут відображаються відгуки, що допомагають оцінити автомобіль перед бронюванням. Цей екран наведено на рисунку 2.3, а форму створення бронювання, у якій користувач задає дати та додаткові опції, показано на рисунку 2.4. У формі бронювання система розраховує підсумкову вартість з урахуванням вибраного періоду оренди та додаткових послуг. Такий підхід дає змогу користувачу одразу бачити фінальну суму й уникати ручного уточнення ціни перед підтвердженням бронювання.

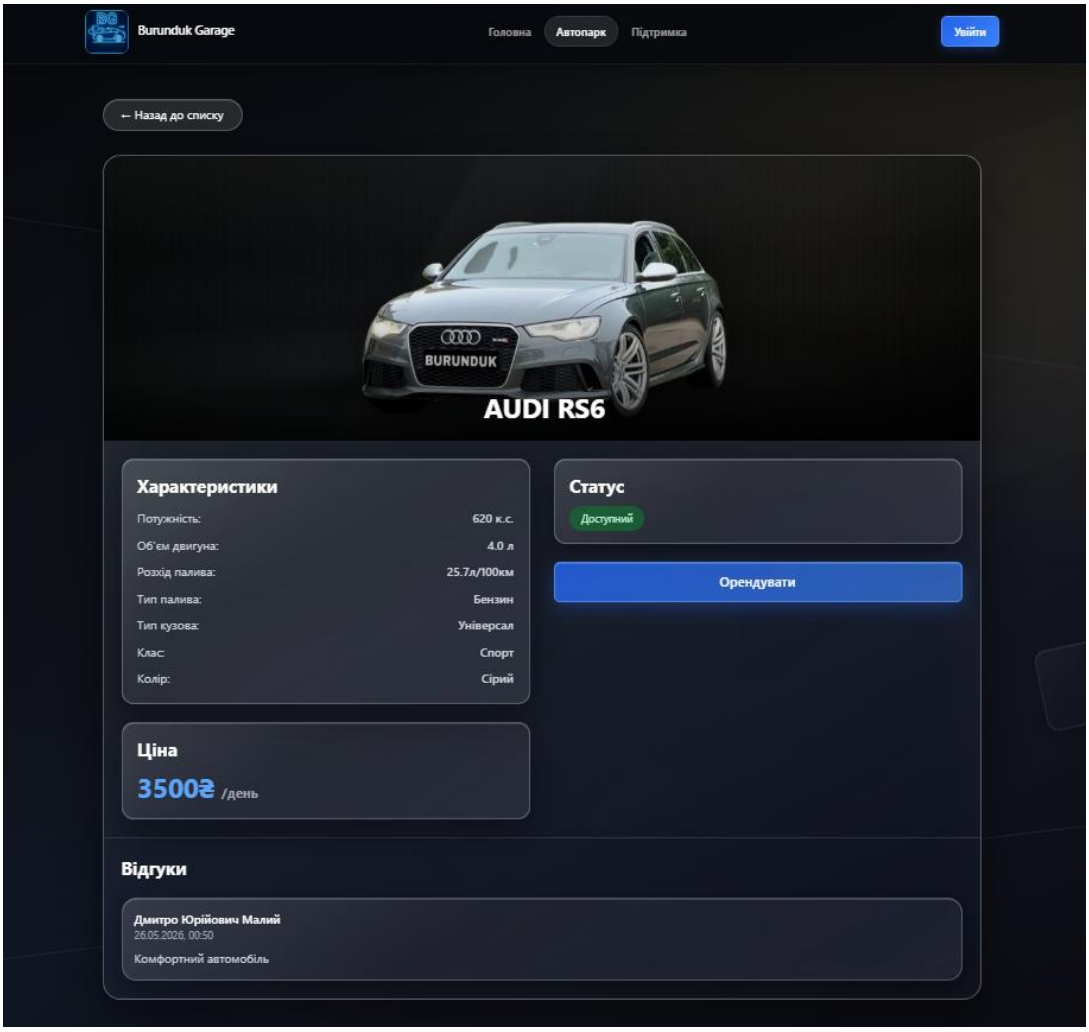


Рисунок 2.3 – Сторінка детального перегляду автомобіля

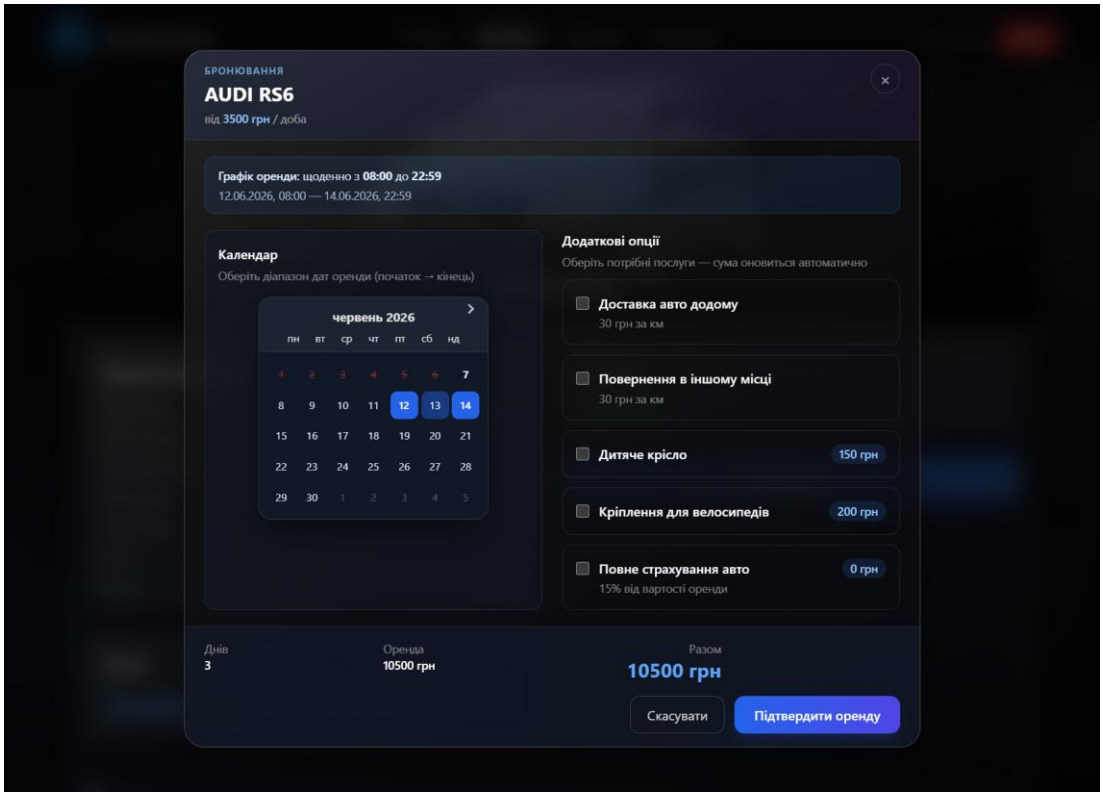


Рисунок 2.4 – Форма створення бронювання

Після оформлення бронювання клієнт працює з розділом «Мої оренди», де може переглядати активні записи, змінювати доступні параметри або скасовувати бронювання у дозволений строк. Цю сторінку наведено на рисунку 2.5.

Для адміністратора створено окремі сторінки керування бронюваннями, користувачами, статистикою та автомобілями, що показано на рисунках 2.6-2.9. У розділі бронювань адміністратор може шукати записи за клієнтом або автомобілем, фільтрувати їх за датами оренди та статусом, а також змінювати додаткові опції. Сторінка користувачів дає змогу швидко знаходити клієнтів за різними даними та редагувати службову інформацію. Панель статистики використовується для перегляду загальних показників роботи сервісу, зокрема кількості бронювань, доходу по місяцях і популярності автомобілів. Окрема сторінка керування автомобілями потрібна для додавання нових авто, зміни характеристик, оновлення фотографій і налаштування позиції зображення в картках автопарку.

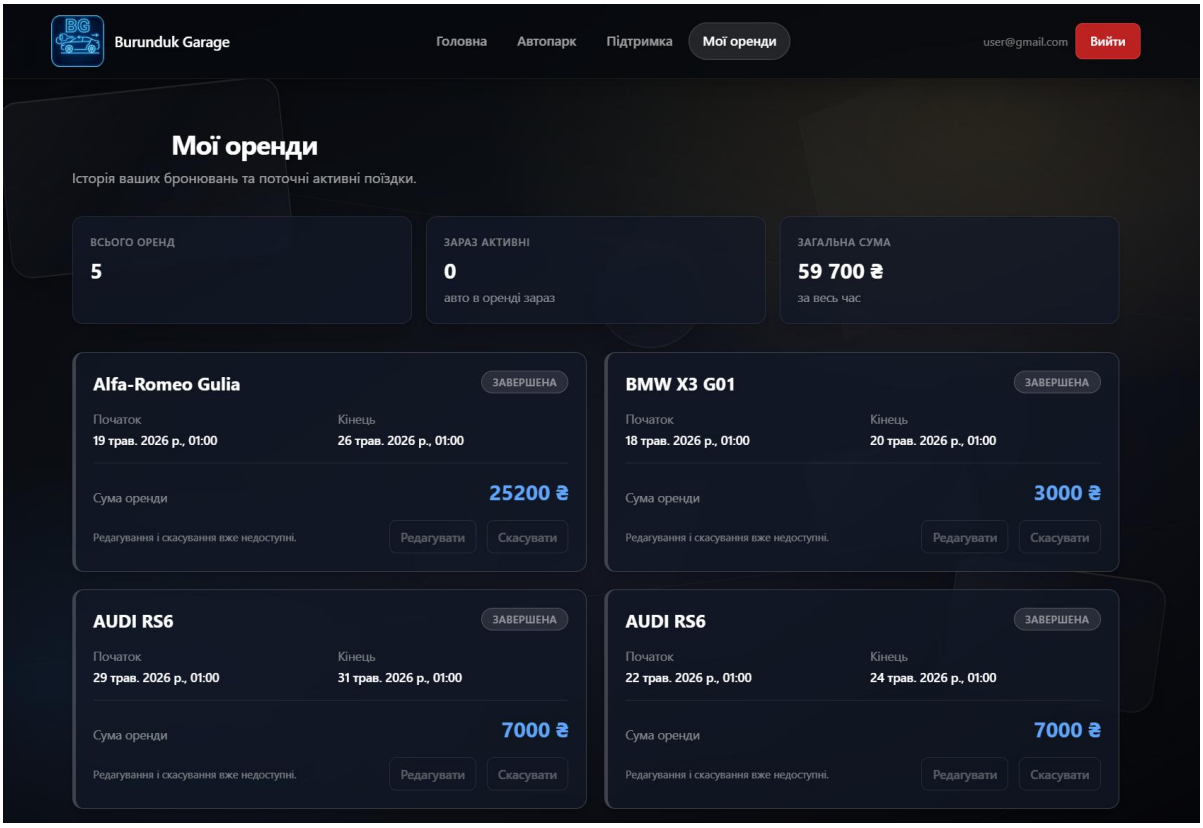


Рисунок 2.5 – Панель користувацьких оренд

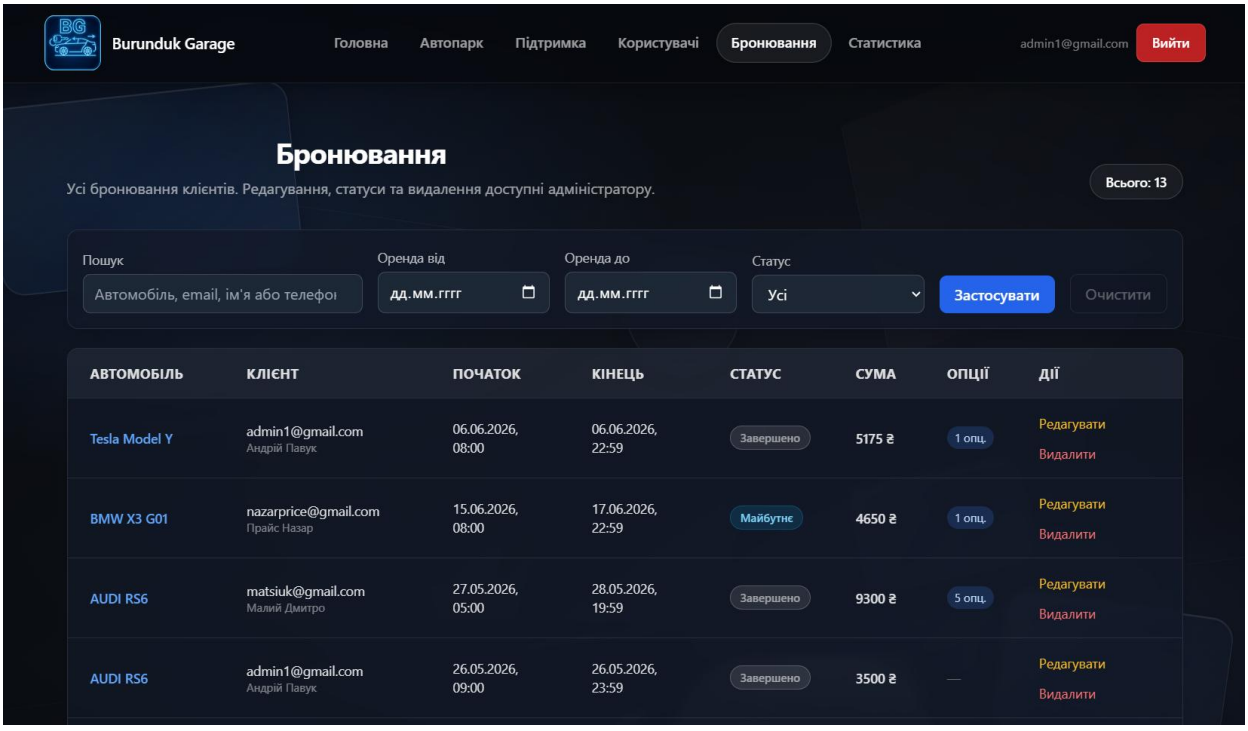


Рисунок 2.6 – Адміністративна сторінка бронювань

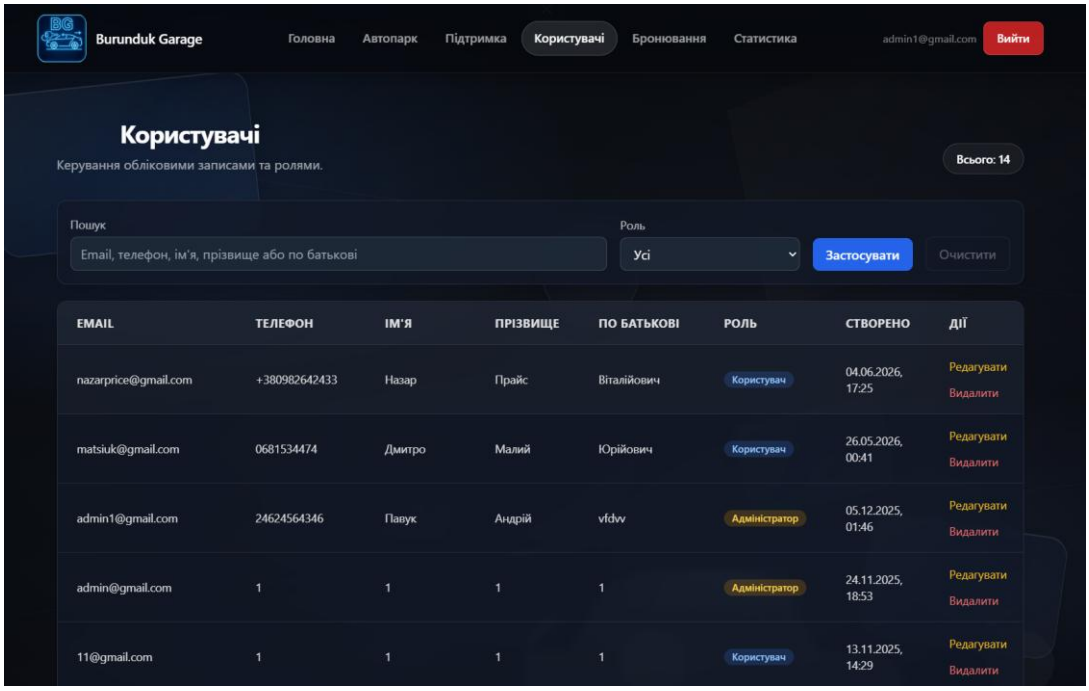


Рисунок 2.7 – Адміністративна сторінка користувачів

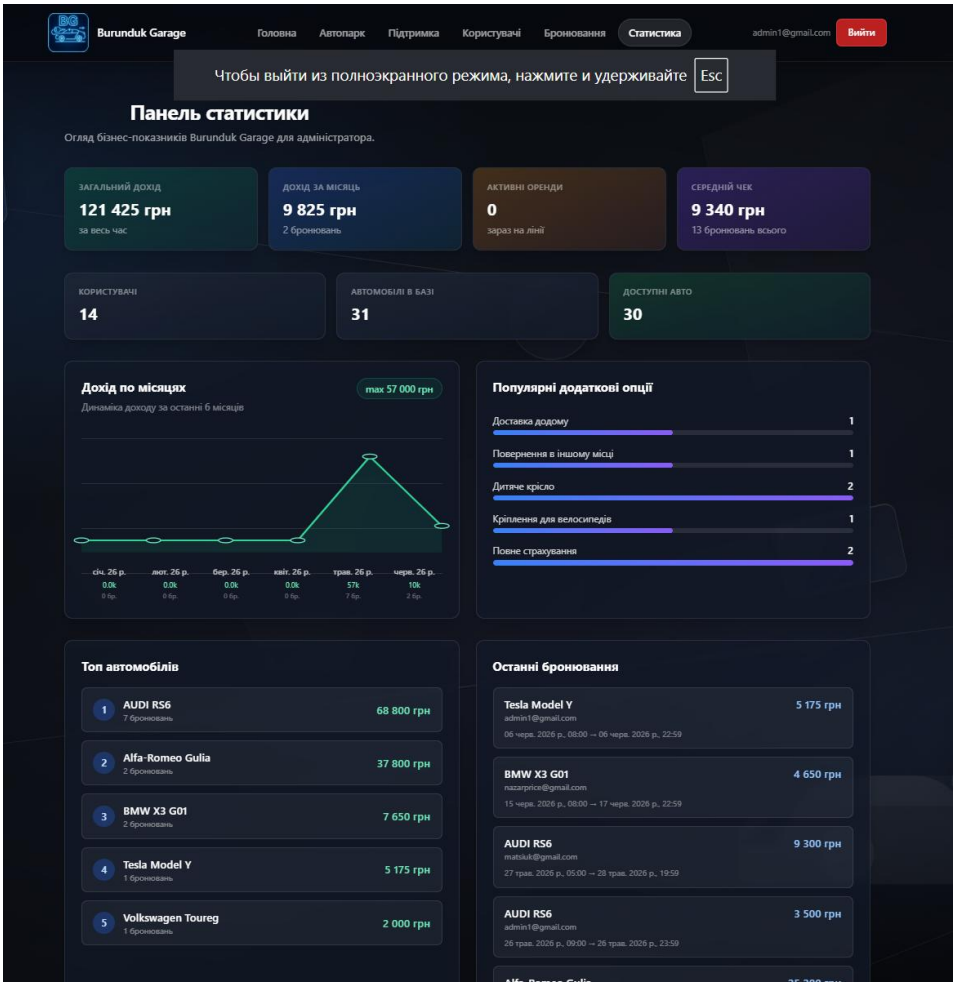


Рисунок 2.8 – Панель статистики адміністратора

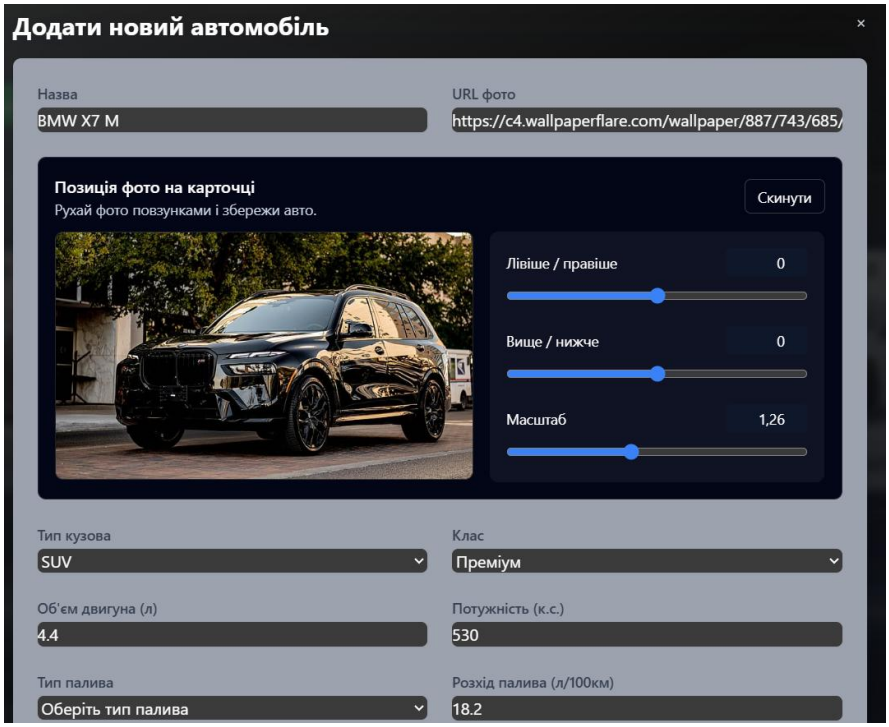


Рисунок 2.9 – Модульне вікно додавання авто

Структура сайту Burunduk Garage побудована так, щоб користувач міг рухатися від загального перегляду автопарку до конкретної дії - бронювання автомобіля. Головна сторінка виконує роль точки входу, автопарк дає змогу порівняти варіанти, а сторінка деталей містить достатньо інформації для прийняття рішення.

Клієнтові потрібен короткий шлях до бронювання. Тому сторінка автомобіля містить фото, характеристики, вартість, дати оренди та кнопку оформлення. Після створення бронювання користувач переходить у панель «Мої оренди», де бачить результат своєї дії.

Службова структура сайту побудована навколо керування даними. Сторінка автомобілів відповідає за каталог, сторінка бронювань - за оренди, сторінка користувачів - за клієнтів, сторінка статистики - за узагальнення діяльності. Поділ на такі сторінки зменшує перевантаження інтерфейсу.

Адміністративні сторінки мають бути доступні тільки після авторизації. На клієнті це забезпечується приватними маршрутами, а на сервері - перевіркою JWT-токена і ролі. Такий подвійний контроль потрібний, тому що приховування

посилання в інтерфейсі саме по собі не є достатнім захистом.

Під час верстки довелося кілька разів повертатися до мобільного вигляду. На комп'ютері користувач бачить ширшу сітку автомобілів і більше простору для характеристик, а на телефоні картки перебудовуються так, щоб фото, назва, клас і ціна не накладалися одне на одного. Саме мобільний автопарк виявився найбільш чутливим до дрібних змін у CSS.

2.2 Створення та верстка сторінок сайту

Створення сторінок сайту виконано за компонентним підходом React. Кожна сторінка складається з окремих компонентів, які відповідають за навігацію, фон, список автомобілів, форму фільтрації, картку автомобіля, модальні вікна, панелі адміністратора та особистий кабінет користувача.

Верстка сторінок побудована з використанням CSS, Tailwind CSS, Flexbox і Grid. Основна увага приділялася адаптивності, щоб автопарк, картки автомобілів, таблиці бронювань і форми коректно відображалися на всіх пристроях.

Для сторінки автопарку реалізовано карточки з матовим фоном, фото автомобіля, короткими характеристиками, ціною і переходом до деталей. Для адміністративних сторінок використано більш щільну структуру: таблиці, фільтри, пошук, кнопки редагування та статуси.

Під час створення інтерфейсу всі сторінки було розподілено за призначенням: частина з них обслуговує клієнтський сценарій, а частина потрібна для адміністрування. Щоб узагальнити структуру інтерфейсу, основні сторінки web-сайту та їхні особливості наведено у таблиці 2.1.

Таблиця 2.1 – Основні сторінки web-сайту

Сторінка	Призначення	Особливості верстки
1	2	3
Головна	Перший екран і перехід до автопарку	Адаптивна навігація, візуальний фон

Продовження таблиці 2.1

1	2	3
Автопарк	Перегляд, фільтрація і сортування автомобілів	Сітка карток, мобільна адаптація
Деталі автомобіля	Характеристики, фото, дати, бронювання	Дві інформаційні зони, форма вибору дат
Мої оренди	Керування власними бронюваннями	Картки оренд і модальне редагування
Admin-панель	Керування автомобілями, користувачами, бронюваннями	Таблиці, пошук, фільтри, форми

Оскільки сторінка автопарку є однією з основних для користувача, її верстка перевірялася не лише на екрані комп'ютера, а й на мобільних пристроях. На малих екранах картки повинні зберігати читабельність, не втрачати ключові характеристики автомобіля і не створювати горизонтального прокручування. Приклад адаптивного відображення сторінки автопарку наведено на рисунку 2.10.

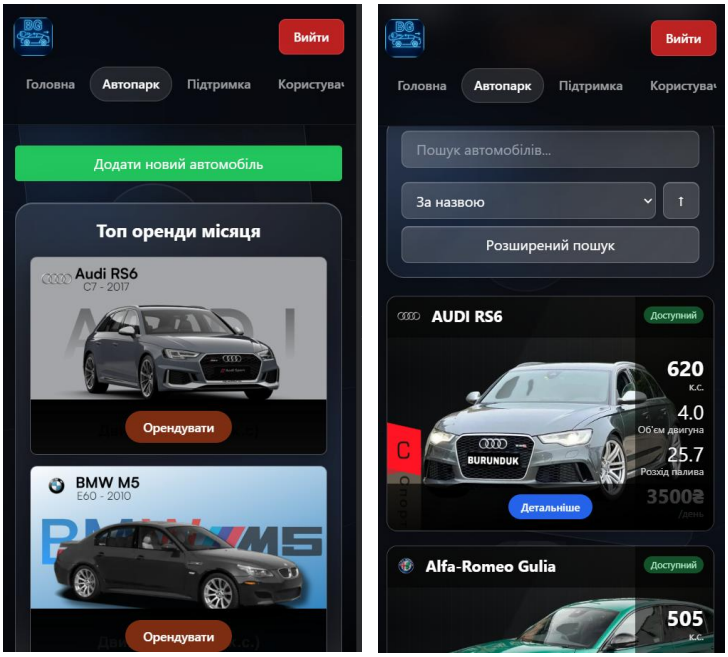


Рисунок 2.10 – Адаптивна верстка сторінки автопарку

Верстка сторінок виконувалася з урахуванням того, що сайт має виглядати не як службова таблиця, а як сучасний сервіс оренди автомобілів. Для карток автопарку використано візуальний акцент на фото автомобіля, короткі характеристики та ціну, щоб користувач міг швидко порівняти варіанти.

Під час створення сторінки автопарку важливо було забезпечити стабільні розміри карток. Якщо текст характеристики стає довшим, наприклад замість «1 опц.» з'являється інший підпис, він не повинен ламати сітку або переноситися так, щоб зміщувати інші елементи. Для цього використовуються фіксовані зони, керовані відступи та адаптивні обмеження.

Сторінка деталей автомобіля поділена на два змістові блоки: візуальне представлення автомобіля та інформацію для бронювання. Завдяки цьому користувач не губиться в характеристиках, але водночас бачить усі дані, які потрібні перед оформленням оренди.

Сторінка «Мої оренди» має більш службовий характер. Тут важливо не декоративне оформлення, а зрозумілі статуси, дати, суми та доступні дії. Кнопки редагування або скасування повинні з'являтися тільки тоді, коли це дозволено правилами системи.

Admin-сторінки верстаються щільніше, ніж клієнтські. У них багато таблиць, фільтрів, полів пошуку та кнопок. Тому дизайн має бути стриманим і передбачуваним: адміністратор повинен швидко сканувати інформацію і виконувати повторювані операції без зайвих переходів.

Під час розробки карток автомобілів виникла практична проблема з фотографіями різних пропорцій. Частина зображень виглядала нормально на широкому екрані, але на мобільному починала обрізатися або зміщувати інформаційний блок. Через це було додано механізм позиціонування фотографії за координатами X та Y і окремий параметр масштабу.

Ще одна дрібна, але помітна проблема з'явилася після зміни підписів характеристик. Довший текст у блоці опцій переносився на новий рядок і ламав компактний вигляд картки. Це довелося виправляти не зміною даних, а саме

версткою: обмеженням ширини, відступів і поведінки тексту всередині картки.

2.3 Розробка структури бази даних сайту

База даних сайту організована у вигляді колекцій MongoDB. Для web-сайту оренди автомобілів це зручно, бо характеристики автомобілів, користувачі, бронювання, відгуки та довідники зберігаються як окремі документи, які можна поступово розширювати.

Для доступу до бази даних використовується Mongoose. У моделях описано структуру документів, типи полів, зв'язки між користувачами, автомобілями та бронюваннями, а також службові поля, наприклад дату створення або дату останньої зміни. Лістинг моделей бази даних наведено в додатку А.

Центральними сутностями є User, Car і Rental. User відповідає за обліковий запис і роль користувача. Car описує автомобіль у каталозі, включно з фото та параметрами позиції зображення. Rental зберігає факт бронювання, дати оренди, додаткові опції, статус і суму.

Після опису основних сутностей потрібно уточнити, які саме колекції використовуються в базі даних і за які дані вони відповідають. Перелік основних колекцій MongoDB, їх призначення та ключові поля наведено у таблиці 2.2.

Таблиця 2.2 – Основні колекції бази даних

Колекція	Призначення	Ключові поля
1	2	3
users	Облікові записи клієнтів і адміністраторів	email, password, role, phone_number, first_name
cars	Автомобілі автопарку	name, body_type, class, fuel_type, price_per_day, photo, image_position_x
rentals	Бронювання та історія оренд	client_id, car_id, start_date, end_date, options, total_price, status

Продовження таблиці 2.2

1	2	3
reviews	Відгуки користувачів про автомобілі	client, car, comment, review_date
statuses	Довідник статусів доступності	status

Окремо важливо показати не лише самі колекції, а й логіку зв'язків між ними. Саме ці зв'язки визначають, як користувачі створюють бронювання, як автомобілі пов'язуються з орендами та як формуються відгуки. Основні зв'язки між сутностями наведено у таблиці 2.3.

Таблиця 2.3 – Зв'язки між сутностями

Зв'язок	**Опис**
User - Rental	Один користувач може мати багато бронювань
Car - Rental	Один автомобіль може мати багато бронювань у різні періоди
Car - Review	Автомобіль може мати багато відгуків
FuelType - Car	Тип пального використовується у багатьох автомобілях
Status - Car	Статус визначає доступність автомобіля

Після визначення основних колекцій та зв'язків між ними логічну модель бази даних доцільно подати у вигляді схеми. На ній видно, що користувач пов'язаний із бронюваннями та відгуками, автомобіль має довідникові характеристики, а кожне бронювання належить одночасно користувачу і конкретному автомобілю. Логічну структуру бази даних сайту зображено у графічній частині 2026.КВР.122.421.02.00.00. БД.

Структура бази даних була спроєктована навколо фактичних процесів автопрокату. Користувач створює бронювання, бронювання пов'язується з автомобілем, а автомобіль має характеристики, статус, фото та довідникові поля. Завдяки цьому в базі можна зберігати не тільки каталог, а й історію взаємодії клієнтів із сервісом.

Модель User містить дані для авторизації та ідентифікації клієнта. Для безпеки пароль не повинен зберігатися у відкритому вигляді, тому на сервері використовується хешування. Роль користувача визначає, які сторінки та API-операції доступні після входу.

Модель Car описує автомобіль як одиницю автопарку. Крім назви, класу, типу кузова, об'єму двигуна, типу пального, ціни та фото, вона містить параметри позиціонування зображення. Це дає адміністратору змогу налаштовувати вигляд картки без додаткового редагування самої фотографії.

Модель Rental є ключовою для логіки оренди. Вона містить посилання на клієнта й автомобіль, дату початку, дату завершення, базову ціну, додаткові опції, статус і загальну суму. Саме на основі цієї моделі виконується перевірка доступності автомобіля на вибраний період.

Довідникові моделі BodyType, Class, FuelType і Status дозволяють підтримувати узгоджені значення в інтерфейсі. Це спрощує фільтрацію та зменшує ймовірність того, що одна й та сама характеристика буде записана різними способами.

2.4 Програмування сайту

Програмування сайту виконано як розробка клієнт-серверного web-застосунку. Клієнтська частина відповідає за інтерфейс, маршрутизацію, відображення даних і взаємодію з користувачем. Серверна частина відповідає за API, авторизацію, перевірку ролей, роботу з MongoDB, бронювання та інтеграцію із зовнішніми сервісами.

Обмін даними між клієнтом і сервером здійснюється через REST API. Клієнт

надсилає HTTP-запити, сервер перевіряє токен, обробляє дані, звертається до бази даних і повертає JSON-відповідь. Це дозволяє розділити інтерфейс і бізнес-логіку.

Окремо реалізовано інтеграцію з ImageKit. Під час роботи із зображеннями автомобілів система використовує посилання на фото і трансформацію видалення фону, а в базі зберігає URL, який потім автоматично відображається на сайті.

Для взаємодії клієнтської частини з сервером реалізовано набір API-маршрутів. Кожен маршрут відповідає за окрему групу дій: роботу з автомобілями, бронюваннями, користувачами, відгуками або статистикою. Основні API-маршрути web-сайту наведено у таблиці 2.4.

Таблиця 2.4 – Основні API-маршрути web-сайту

Маршрут	**Метод**	**Призначення**
/api/cars	GET/POST	Отримання списку автомобілів і створення нового запису
/api/cars/:id	GET/PUT/DELETE	Перегляд, редагування та видалення автомобіля
/api/cars/:id/check-availability	POST	Перевірка доступності автомобіля за датами
/api/rentals	GET/POST	Отримання і створення бронювань
/api/rentals/:id	PUT/DELETE	Редагування або скасування бронювання
/api/users	GET	Отримання користувачів для адміністративної сторінки
/api/admin/statistics	GET	Отримання статистичних показників

Оскільки web-сайт складається з клієнтської частини, серверної частини, бази

даних і зовнішнього сервісу ImageKit, для пояснення обміну даними між ними використано окрему схему. Вона показує, як React-клієнт надсилає запити до Express-сервера, сервер працює з MongoDB, а фотографії автомобілів обробляються через ImageKit. Схему програмної взаємодії частин web-сайту подано в додатку Н.

Програмування сайту виконувалося з поділом на frontend і backend. Такий поділ дозволяє незалежно розвивати інтерфейс і серверну логіку. Frontend відповідає за взаємодію з користувачем, а backend - за перевірки, збереження даних, авторизацію та бізнес-правила.

На сервері Express реалізовано API-маршрути для автомобілів, бронювань, користувачів, статистики, відгуків і довідників. Кожен маршрут має свою роль: частина доступна всім користувачам, частина потребує авторизації, а адміністративні операції додатково перевіряють роль admin. Фрагмент серверної частини Express наведено в додатку Б.

Під час бронювання сервер не повинен довіряти лише клієнтському інтерфейсу. Навіть якщо на фронтенді кнопка прихована, сервер повинен повторно перевіряти права користувача, дату початку оренди, перетини з іншими бронюваннями та коректність вхідних даних. Послідовність перевірок під час створення бронювання подано у вигляді блок-схеми в додатку М.

ImageKit використовується для роботи із зображеннями автомобілів. Якщо адміністратор додає посилання на фото, сервер може зберегти оброблене посилання з трансформацією видалення фону. Це спрощує підтримку єдиного стилю карток автомобілів. Лістинг модуля ImageKit наведено в додатку В.

Програмування також включало оптимізацію дрібних користувацьких сценаріїв. Наприклад, пошук у таблицях не повинен перезавантажувати сторінку після введення однієї літери, а сортування за розходом палива повинно порівнювати числа, навіть якщо значення прийшло з бази у вигляді рядка.

Під час реалізації фільтрації бронювань виникла проблема з обробкою дат між клієнтською та серверною частинами. На клієнті дата вибиралася як об'єкт або

рядок у локальному форматі, а MongoDB очікувала коректний часовий проміжок для запиту. Для виправлення було узгоджено формат передачі параметрів і додано обробку дат на сервері перед формуванням умови пошуку.

Окремо довелося перевіряти сортування за розходом палива. Спочатку значення поводилися як рядки, тому автомобіль із дробовим числом міг стояти після автомобіля з більшим цілим значенням. Після цього сортування було змінено так, щоб перед порівнянням витягувати числову частину характеристики.

З ImageKit теж виникла практична особливість: локально фото могло виглядати правильно, але після публікації треба було переконатися, що зберігається саме URL з трансформацією видалення фону, а не початкове посилання. Через це логіку роботи із зображенням було винесено окремо й перевірено як для створення, так і для редагування автомобіля.

2.4.1 Написання клієнтської частини

Клієнтська частина web-сайту реалізована на React. Вона містить сторінки головного екрана, автопарку, деталей автомобіля, входу, реєстрації, підтримки та панель «Мої оренди». Навігація між сторінками виконується через React Router.

Для роботи з даними клієнтська частина використовує сервіс API, який централізує HTTP-запити до сервера. Це зменшує дублювання коду і спрощує обробку помилок. Окремі утиліти відповідають за форматування характеристик автомобілів, розрахунок вартості оренди та відображення інформації залежно від типу пального. Лістинг сервісу API наведено в додатку Г, а лістинг форматування характеристик автомобіля наведено в додатку К.

У клієнтській частині реалізовано фільтри автопарку, сортування за розходом палива, сторінку деталей автомобіля, модальне вікно бронювання, редагування власної оренди та скасування бронювання за два дні до початку оренди. Лістинг компонента автопарку наведено в додатку Д, а лістинг розрахунку вартості оренди наведено в додатку Л.

Клієнтська частина реалізована на React, що дозволяє будувати інтерфейс із

повторно використовуваних компонентів. Компоненти автопарку, форми фільтрації, модальних вікон, сторінки деталей, header і сторінки оренд працюють як окремі частини, але разом формують єдиний сценарій користувача.

React Router використовується для переходів між сторінками без повного перезавантаження web-сайту. Це робить роботу плавнішою, особливо під час переходу від автопарку до деталей автомобіля або від оформлення бронювання до панелі «Мої оренди».

Сервіс API у клієнтській частині централізує запити до backend. Завдяки цьому компоненти не дублюють адресу сервера, заголовки авторизації та обробку помилок. Якщо змінюється адреса backend після розгортання, достатньо налаштувати змінну середовища VITE_API_URL.

Відображення автомобілів спирається на допоміжні функції, які визначають підписи характеристик. Це дозволяє коректно показувати автомобілі з альтернативними характеристиками: ті самі поля бази можуть відображатися як ємність батареї та витрата електроенергії, не вимагаючи зміни структури backend.

Клієнтська частина також відповідає за попередню зручність користування. Наприклад, вона показує, чи доступне редагування оренди, відкриває модальне вікно зміни дат, відображає помилки запиту та оновлює список бронювань після успішної дії. Фрагмент сторінки «Мої оренди» наведено в додатку И.

2.4.2 Написання admin- частини

Admin-частина web-сайту доступна тільки користувачам із роллю адміністратора. Захист реалізовано через приватні маршрути на клієнті та перевірку JWT-токена і ролі на сервері. Це не дозволяє звичайному користувачу відкрити службові сторінки через пряме посилання.

В admin-частині реалізовано керування автомобілями, користувачами, бронюваннями і статистикою. Адміністратор може додавати та редагувати автомобілі, завантажувати фотографії через ImageKit, змінювати положення фото у картці, переглядати бронювання, шукати записи за клієнтом або автомобілем і

фільтрувати їх за датами.

Сторінка статистики показує загальні показники роботи сервісу, зокрема кількість бронювань, автомобілів, користувачів і дохід по місяцях. Для зручності адміністратора використано таблиці, фільтри, статуси, модальні вікна редагування та окремі форми.

Admin-частина створена для користувача, який керує автопарком і бронюваннями. Вона відрізняється від клієнтської частини тим, що має більше таблиць, фільтрів і форм. Її головна мета - швидке виконання службових операцій, а не презентаційний вигляд.

Сторінка керування автомобілями дозволяє додавати нові записи, редагувати характеристики, змінювати фото тналаштовувати позицію зображення. Повзунки X, Y і масштаб дають можливість підігнати фото автомобіля до картки без ручного обрізання в окремому редакторі. Фрагмент форми автомобіля наведено в додатку Е.

Сторінка бронювань підтримує пошук за клієнтом або автомобілем. Це важливо, тому що адміністратор може отримати звернення від клієнта і швидко знайти його оренду, навіть якщо не пам'ятає точну дату. Фільтрація за датами дозволяє працювати з конкретним періодом. Фрагмент адміністративної сторінки бронювань наведено в додатку Ж.

Сторінка користувачів реалізує пошук за різними даними користувача. Це може бути ім'я, прізвище, email або телефон. Такий пошук зменшує час на адміністрування, особливо коли база клієнтів поступово зростає.

Сторінка статистики призначена для швидкої оцінки роботи сервісу. Вона показує загальний дохід, кількість бронювань, активні оренди, нових користувачів і дохід по місяцях. У результаті стан автопарку видно без ручних підрахунків у таблицях.

2.5 Тестування web- сайту

Тестування web-сайту виконувалося за сценаріями, що відповідають реальній

роботі користувача й адміністратора. Основну увагу приділено перевірці авторизації, правильності роботи фільтрів, створенню та редагуванню бронювань, відображенню автомобілів і адаптивності інтерфейсу.

Під час налагодження перевірялися помилки, які можуть виникати при роботі з датами, пошуком, фільтрацією та сортуванням. Наприклад, окремо перевірялася коректність сортування автомобілів за розходом палива, оскільки числові значення можуть зберігатися як рядки та потребують правильного перетворення перед порівнянням.

Для адміністративної сторінки бронювань перевірялося, що введення символів у пошук не призводить до повного перезавантаження сторінки, фільтри за датами не створюють помилку отримання оренд, а варіанти статусу включають майбутні та завершені бронювання.

Для мобільної адаптації перевірялися картки автомобілів, відступи, тіні, положення значків класу, ширина сторінок і коректність відображення основних елементів. Для десктопної версії додатково перевірялася плавність прокручування, оскільки надмірні декоративні ефекти можуть впливати на продуктивність.

Тестування web-сайту виконувалося не тільки через перевірку окремих кнопок, а й через проходження повних сценаріїв. Наприклад, користувач реєструється, входить у систему, відкриває автопарк, фільтрує автомобілі, обирає один із них, задає дати та створює бронювання.

Для admin-частини тестувалися пошук і фільтри. Важливо було переконатися, що введення в поле пошуку не призводить до перезавантаження сторінки, а вибір дати у фільтрі бронювань не викликає помилки отримання даних із сервера.

Сортування за розходом палива перевірялося окремо, оскільки такі значення можуть містити дробову частину. Якщо порівнювати їх як рядки, порядок буде неправильним. Тому під час тестування перевірялося, що значення перетворюються на числа перед сортуванням.

Також перевірялася адаптивність на мобільних екранах. Картки автомобілів

не повинні ламати структуру, значки класу мають залишатися на правильному місці, а текст характеристик не повинен накладатися на фото або ціну. Це особливо важливо для сторінки автопарку.

Щоб перевірити працездатність web-застосунку, були сформовані тестові сценарії для користувацької та адміністративної частин. Вони охоплюють авторизацію, бронювання, редагування оренди, роботу фільтрів, адаптивність і розгортання. Перелік тестових сценаріїв наведено у таблиці 2.5.

Таблиця 2.5 – Перелік тестових сценаріїв

Сценарій	**Очікуваний результат**	**Статус**
Реєстрація та вхід користувача	Користувач отримує доступ до особистих сторінок	Виконано
Створення бронювання	Бронювання зберігається, якщо автомобіль доступний	Виконано
Редагування оренди користувачем	Зміна доступна лише за два дні до початку	Виконано
Пошук на сторінці бронювань	Сторінка не перезавантажується під час введення	Виконано
Фільтр бронювань за датами	Повертаються записи у вибраному проміжку	Виконано
Сортування за розходом палива	Числові значення сортуються коректно	Виконано
Редактор фотографії авто	Зміни позиції та масштабу зберігаються	Виконано
Мобільна адаптація автопарку	Картки не виходять за межі екрана	Виконано

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з розміщення сайту в Інтернеті

Інструкція з розміщення сайту в Інтернеті описує підготовку клієнтської та серверної частини до публікації. Клієнтська частина після збірки формує статичні файли, які можуть бути розміщені на хостингу для фронтенду. Серверна частина розміщується на платформі, що підтримує Node.js і змінні середовища.

Перед розгортанням потрібно налаштувати підключення до MongoDB, ключі JWT, параметри ImageKit і адресу API для клієнтської частини. Також необхідно переконатися, що сервер дозволяє CORS-запити з домену фронтенду, а всі секретні дані зберігаються не в коді, а в конфігурації середовища.

Після публікації потрібно перевірити головну сторінку, автопарк, авторизацію, сторінку деталей автомобіля, створення бронювання, адміністративний вхід і завантаження зображень. Особливу увагу слід приділити тому, що локальні шляхи до іконок або файлів можуть працювати на локальному сервері, але не працювати після розгортання, якщо вони неправильно підключені.

Розміщення проєкту в Інтернеті виконується не як один монолітний сайт, а як кілька пов'язаних сервісів. Frontend публікується на Vercel, backend запускається на Render, MongoDB використовується для збереження даних, а ImageKit відповідає за зберігання та обробку зображень.

Проєкт Burunduk Garage розгортається як дві окремі частини: frontend на Vercel і backend на Render. Такий поділ відповідає клієнт-серверній архітектурі: Vercel обслуговує React-застосунок, а Render запускає Node.js сервер з Express і підключенням до MongoDB.

Для розміщення frontend на Vercel потрібно підготувати репозиторій із клієнтською частиною, підключити його в панелі Vercel, вибрати фреймворк Vite, вказати команду збірки npm run build і директорію результату dist. Після цього Vercel автоматично збирає сайт і видає публічне посилання.

У змінних середовища Vercel потрібно задати адресу backend. Для цього

використовується VITE_API_URL, у якій записується URL серверної частини на Render з додаванням шляху /api. Це потрібно, щоб після розгортання клієнтська частина надсилала запити не на localhost, а на реальний сервер.

Backend розміщується на Render як Web Service. У налаштуваннях сервісу потрібно вибрати репозиторій, середовище Node.js, команду встановлення залежностей npm install і команду запуску npm start. Якщо backend знаходиться в окремій папці, у Render потрібно правильно вказати root directory.

Для backend на Render обов'язково задаються змінні середовища: рядок підключення до MongoDB, секрет JWT, ключі ImageKit, URL endpoint ImageKit і порт, якщо він потрібний у конфігурації. Секретні ключі не повинні зберігатися у відкритому коді репозиторію.

Після запуску backend потрібно перевірити базовий маршрут сервера та кілька API-запитів, наприклад отримання списку автомобілів або авторизацію. Якщо сервер відповідає, можна повертатися до Vercel і перевіряти, чи frontend правильно отримує дані через VITE_API_URL.

Окремо необхідно налаштувати CORS на backend. Сервер повинен дозволяти запити з домену Vercel, інакше браузер заблокує звернення клієнтської частини до API. На етапі тестування можна дозволити кілька адрес, але для фінального розгортання бажано залишити тільки потрібні домени.

Після розміщення обох частин виконується контрольна перевірка: відкриття головної сторінки, перегляд автопарку, вхід користувача, створення бронювання, відкриття admin-сторінок, робота з ImageKit і відображення статистики. Якщо всі сценарії проходять успішно, сайт можна вважати опублікованим.

Перед розгортанням frontend потрібно переконатися, що локальна збірка виконується без помилок. Для цього у папці клієнтської частини запускається встановлення залежностей і команда збірки. Якщо Vite успішно формує папку dist, проєкт готовий до публікації на Vercel.

У Vercel доцільно підключати саме той репозиторій, у якому зберігається актуальна версія frontend. Після кожного push у вибрану гілку Vercel може

автоматично запускати новий deployment. Це зручно, оскільки зміни в інтерфейсі автопарку або сторінках бронювань одразу потрапляють у тестове чи робоче середовище.

Для backend на Render важливо перевірити, що сервер слухає порт із середовища, а не тільки фіксоване локальне значення. На Render порт передається платформою, тому сервер Express повинен використовувати process.env.PORT з резервним значенням для локального запуску.

MongoDB краще використовувати як окремий хмарний сервіс, наприклад MongoDB Atlas. У Render зберігається тільки рядок підключення, а сама база даних залишається незалежною від сервера. Це спрощує перенесення backend і зменшує ризик втрати даних під час оновлення сервісу.

Після першого розгортання потрібно перевірити CORS у реальних умовах. Якщо frontend відкривається, але не завантажує автомобілі, причина часто полягає у неправильній адресі API або забороні запитів із домену Vercel. У такому випадку необхідно оновити список дозволених origin на backend.

Також потрібно перевірити роботу ImageKit після розгортання. Локально URL може формуватися правильно, але на сервері мають бути задані всі ключі ImageKit. Якщо хоча б один ключ відсутній, додавання або оновлення фото автомобіля може завершуватися помилкою.

Під час перенесення проєкту з локального середовища на Vercel і Render потрібно окремо перевіряти не тільки запуск, а й фактичні запити між частинами системи. Найчастіше помилки виникають не в React-компонентах, а в адресі API, CORS або відсутній змінній середовища на Render.

Для цього після кожного розгортання варто пройти короткий контрольний маршрут: відкрити автопарк, авторизуватися, створити тестове бронювання, зайти в admin-частину і перевірити сторінку бронювань. Така перевірка швидко показує, чи правильно frontend на Vercel звертається до backend на Render.

3.2 Інструкція з обслуговування та наповнення сайту

Адміністратор входить у систему під обліковим записом із відповідною роллю. Після входу йому доступні сторінки керування автомобілями, бронюваннями, користувачами і статистикою. Якщо користувач не має ролі адміністратора, доступ до цих сторінок має бути заблокований.

Для додавання автомобіля адміністратор заповнює основні поля: марку, модель, рік, ціну, клас, тип кузова, тип пального, характеристики, посилання на фотографію та інші параметри. Якщо автомобіль електричний, у полях об'єму та витрати можуть вноситися дані про батарею і витрату електроенергії, а інтерфейс відобразить їх з відповідними підписами.

Після додавання або редагування зображення адміністратор може скористатися редактором позиції. Повзунки image_position_x, image_position_y та image_zoom дозволяють змістити фотографію праворуч, ліворуч, вгору, вниз і змінити масштаб так, щоб автомобіль гармонійно виглядав у картці.

На сторінці бронювань адміністратор може знаходити записи за клієнтом або автомобілем, фільтрувати їх за датами оренди та статусом, переглядати майбутні й завершені бронювання, змінювати статус і додаткові опції. Це дає змогу швидко обробляти поточні звернення клієнтів.

На сторінці користувачів адміністратор використовує пошук за різними даними. Пошук не повинен перезавантажувати сторінку під час введення, тому він працює як фільтрація даних або запит із контрольованим станом форми.

Обслуговування сайту Burunduk Garage починається з контролю актуальності даних. Адміністратор повинен регулярно перевіряти список автомобілів, ціни, статуси, фотографії та характеристики. Якщо автомобіль тимчасово недоступний, це має бути відображено в системі.

Під час додавання нового автомобіля адміністратор заповнює основні характеристики, вибирає довідникові значення та додає фото. Якщо фото завантажується через ImageKit або вже містить трансформацію видалення фону, у картці воно відображається в єдиному стилі з іншими автомобілями.

Після додавання фото бажано перевірити його положення на сторінці автопарку. Якщо автомобіль занадто зміщений, обрізаний або виглядає непропорційно, адміністратор може змінити image_position_x, image_position_y та image_zoom через редактор у формі автомобіля.

Обслуговування бронювань включає перегляд нових оренд, перевірку майбутніх бронювань і зміну статусу в разі скасування. Для швидкої роботи адміністратор використовує пошук за клієнтом або автомобілем, а також фільтри за датами.

Сторінка користувачів використовується для перегляду клієнтської бази. Пошук за різними даними допомагає знайти потрібного користувача навіть тоді, коли відома лише частина інформації. Це спрощує обробку звернень і уточнення деталей бронювання.

Технічне обслуговування також включає оновлення залежностей, перевірку логів Render, контроль змінних середовища, доступності MongoDB та коректності роботи ImageKit. Після кожного значного оновлення потрібно перевіряти основні сценарії користувача й адміністратора.

Після розміщення сайту адміністратор повинен вести автопарк як актуальний електронний каталог. Якщо автомобіль змінює ціну, статус або фото, ці дані потрібно оновити в admin-панелі до того, як клієнт оформить нове бронювання.

Під час наповнення сайту варто дотримуватися єдиного стилю описів. Назви автомобілів, характеристики, витрата палива та ціна повинні подаватися однаково для всіх карток. Це робить автопарк зручним для порівняння.

Обслуговування бронювань передбачає регулярну перевірку майбутніх оренд. Адміністратор може використовувати фільтр за датами, щоб бачити найближчі видачі автомобілів, або фільтр за завершеними орендами, щоб аналізувати історію роботи сервісу.

Якщо користувач звертається з проханням змінити бронювання, адміністратор може знайти його за іменем клієнта або автомобілем. Після цього перевіряється новий період оренди і, якщо він не конфліктує з іншими

бронюваннями, дані оновлюються в системі.

Для стабільної роботи сайту потрібно періодично перевіряти логи Render. У логах можуть бути помилки підключення до бази даних, неправильні токени, некоректні запити або проблеми з ImageKit. Регулярний перегляд логів допомагає знаходити проблеми до того, як їх помітять клієнти.

3.3 Інструкція з популяризації та підтримки сайту

Після запуску вебзастосунку важливо підтримувати актуальність автопарку. Фотографії, ціни, статуси та характеристики автомобілів повинні оновлюватися одразу після зміни реального стану. Це підвищує довіру клієнтів і зменшує кількість уточнень через телефон або месенджери.

Для популяризації сайту доцільно використовувати соціальні мережі, локальну рекламу, пошукову оптимізацію сторінок автомобілів і посилання на сайт у профілях сервісу. Оскільки клієнти часто обирають автомобіль візуально, особливу увагу потрібно приділяти якісним фото, єдиному стилю карток і зрозумілим характеристикам.

Підтримка вебзастосунку включає резервне копіювання бази даних, оновлення залежностей, контроль помилок сервера, перевірку працездатності ImageKit і періодичне тестування ключових сценаріїв. Якщо додаються нові функції, їх потрібно перевіряти окремо на десктопі та мобільних пристроях.

Подальший розвиток системи може включати онлайн-оплату, службові сповіщення, календар зайнятості автопарку, персональні промокоди, інтеграцію з месенджерами, розширену аналітику та автоматичні нагадування клієнтам перед початком оренди.

Популяризація сайту Burunduk Garage повинна спиратися на зрозумілу демонстрацію автопарку. Клієнт найчастіше приймає рішення за фото, ціною, класом автомобіля та умовами оренди, тому сторінки автомобілів мають бути актуальними й візуально охайними.

Для просування можна використовувати соціальні мережі, локальну рекламу,

посилання в профілях компанії та пошукову оптимізацію сторінок. У текстах варто робити акцент на доступних автомобілях, простому бронюванні, прозорих датах і можливості швидко переглянути варіанти онлайн.

Підтримка сайту після запуску повинна бути регулярною. Якщо змінюються ціни, доступність автомобілів або умови оренди, інформацію потрібно оновлювати одразу. Наявність застарілих даних може створити недовіру до сервісу та збільшити кількість ручних уточнень.

Для розвитку web-сайту можна додати онлайн-оплату, автоматичні повідомлення клієнтам, календар зайнятості автомобілів, промокоди, рейтинг автомобілів і розширену аналітику. Такі функції не є обов'язковими для першої версії, але вони можуть підвищити зручність сервісу в майбутньому.

Окремим напрямом підтримки є контроль продуктивності. Якщо сайт починає працювати повільніше, потрібно перевірити розмір зображень, кількість запитів до API, складність декоративних ефектів і роботу серверних маршрутів. Для сервісу оренди сторінка автопарку має відкриватися швидко і плавно прокручуватися.

Під час популяризації Burunduk Garage доцільно використовувати прямі посилання на сторінку автопарку. Користувач повинен одразу бачити доступні автомобілі, а не шукати каталог через кілька проміжних сторінок. Це скорочує шлях від реклами до бронювання.

Для соціальних мереж можна готувати публікації з окремими автомобілями, вказуючи основні характеристики та посилання на сторінку. Якщо фото в картках мають однаковий стиль завдяки ImageKit, візуальна подача виглядає цілісніше.

Підтримка сайту включає не тільки технічні оновлення, а й аналіз поведінки клієнтів. Якщо певні автомобілі часто переглядають, але рідко бронюють, варто перевірити ціну, фото, опис або доступність дат. Такі спостереження можуть впливати на розвиток автопарку.

У майбутньому сайт можна доповнити сторінками з умовами оренди, відповідями на часті запитання, блоком акцій. Це підвищить інформативність сайту

і може покращити його позиції у пошуку.

Підтримка не повинна порушувати стабільність уже реалізованих функцій. Перед додаванням нових можливостей потрібно перевіряти авторизацію, бронювання, фільтри, роботу admin-панелі та адаптивність. Для сервісу оренди навіть невелика помилка в датах може мати практичні наслідки.

Практична підтримка сайту має виконуватися за простим регламентом. Один раз на тиждень доцільно перевіряти доступність основних сторінок, коректність авторизації, створення бронювання, відкриття панелі «Мої оренди» та роботу адміністративних таблиць. Таку перевірку бажано проводити окремо для десктопної та мобільної версії, оскільки частина користувачів буде відкривати сайт саме з телефона після переходу з реклами або соціальних мереж.

Для популяризації важливо не тільки залучити користувача на сайт, а й швидко привести його до потрібної дії. Тому рекламні матеріали варто спрямовувати безпосередньо на сторінку автопарку або на конкретний автомобіль. У таких матеріалах доцільно вказувати клас авто, ціну за добу, основні характеристики та переваги онлайн-бронювання. Це робить повідомлення конкретним і зменшує кількість зайвих уточнень перед оформленням оренди.

Окремо потрібно стежити за якістю даних у каталозі. Якщо адміністратор додає новий автомобіль, потрібно перевірити назву, тип кузова, клас, тип пального, витрату, ціну, статус і фотографію. Після редагування фото бажано переглянути картку в автопарку та сторінку деталей автомобіля, щоб переконатися, що позиціонування зображення не спотворює вигляд сторінки. Така перевірка особливо важлива для автомобілів із фотографіями різних пропорцій.

Підтримка також повинна враховувати безпеку й відновлення даних. Для цього необхідно зберігати резервні копії бази даних, контролювати доступ до змінних середовища на Render і Vercel, не передавати службові ключі стороннім особам та періодично перевіряти, чи не з'явилися помилки. Якщо після чергового оновлення змінюється поведінка фільтрів, дат або статусів бронювання, такі зміни потрібно виправляти до активного використання сайту клієнтами.

4. ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини даного дипломного проєкту є проведення економічних розрахунків, спрямованих на визначення економічної ефективності розробки вебзастосунку для автоматизації управління автопарком «Burunduk Garage», прийняття рішення про подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки.

Об’єктом розробки є вебзастосунок «Burunduk Garage», призначений для автоматизації процесів оренди автомобілів, ведення електронного каталогу автопарку, обліку замовлень клієнтів, формування адміністративної звітності та підвищення ефективності роботи служби прокату транспортних засобів

Розрахунок вартості розробки виконується в декілька етапів:

– описати технологічний процес розробки із зазначенням трудомісткості кожної операції;

– визначити суму витрат на оплату праці основного і допоміжного персоналу, включаючи відрахування на соціальні заходи;

– обчислити витрати на електроенергію;

– нарахувати суму амортизаційних відрахувань;

– визначити суму накладних витрат;

– скласти кошторис та визначити собівартість робіт;

– розрахувати ціну робіт;

– визначити економічну ефективність та термін окупності.

4.1. Визначення стадій технологічного процесу та загальної тривалості проведення НДР

У цьому підрозділі деталізовано життєвий цикл створення вебзастосунку «Burunduk Garage». Для наочного представлення часових витрат та розподілу обов'язків між членами команди, тривалість кожної окремої стадії технологічного процесу наведено у таблиці 4.1.

Таблиця 4.1 – Часові нормативи та етапи розробки вебплатформи «Burunduk Garage»

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1	Аналітика, дослідження вимог та планування	Кер. проєкту Pm	10
		Інженер (l1)	6
2	Формування та затвердження ТЗ	Кер. проєкту (Pm)	8
		Інженер (l1)	6
3	Формування та затвердження ТЗ	Інженер (l1)	20
		Інженер (l2)	20
4	Написання програмного коду (реалізація функціоналу)	Інженер (l1)	48
5	Проведення тестування, пошук та усунення багів	Тестувальник	12
6	Складання технічної документації	Інженер (l1)	4
7	Деплой, налаштування серверного середовища	Інженер (l2)	14
8	Загальний менеджмент та координація	Кер. проєкту (Pm)	22
Разом			170

Загальна трудомісткість розроблення вебзастосунку становить **170 годин.**

4.2. Визначення витрат на оплату праці та відрахувань на соціальні заходи

Цей розділ присвячено розрахунку фінансових витрат, спрямованих на матеріальну винагороду команди розробників, а також супутніх податкових зобов'язань. Базовий рівень заробітної плати фахівців формується з урахуванням ринкових умов, кваліфікаційного рівня та безпосередньої кількості годин, витрачених на завдання.

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та діяльності підприємства.

Основна заробітна плата розраховується за формулою:

$$З_{осн.} = Т_с * К_г \quad (4.1)$$

де: T_c – тарифна ставка, грн. (приймаємо для керівника проєкту (Pm) – 520 грн./год, інженера (l2) – 340 грн./год., інженера (l1) – 150 грн./год., тестувальник – 130 грн./год.; $K_г$ – кількість відпрацьованих годин.

Проведемо розрахунок базової зарплати за ролями:

Керівника проекту (Pm) $З_{осн2}$= 40 * 520 = 20 800 грн.

Інженера (l2) $З_{осн4}$= 34 * 340 = 11 560 грн.

Інженера (l1) $З_{осн3}$= 84 * 150 = 12 600 грн.

Тестувальник $З_{осн4}$= 12 * 130 = 1560 грн.

Сумарна основна заробітна плата становить

$$З_{осн} = 20\,800 + 11\,560 + 12\,600 + 1560 = 46\,520 \text{ грн.}$$

Додаткова заробітна плата становить 10 – 15 % від суми основної заробітної плати.

$$З_{дод.} = З_{осн.} \cdot К_{допл.} \quad (4.2)$$

де: $К_{допл.}$ – коефіцієнт додаткових виплат працівникам.

Отже додаткова заробітна плата по категоріях працівників становить:

Керівника проекту $З_{дод2}$= 20 800 * 0,1 = 2080 грн.

Інженера (l2) $З_{дод3}$= 11 560 * 0,1 = 1156 грн.

Інженера (l1) $З_{дод4}$= 12 600 * 0,1 = 1260 грн.

Тестувальник $З_{дод4}$= 1560 * 0,1 = 156 грн.

Загальна додаткова заробітна плата становить:

$$З_{дод} = 2080 + 1156 + 1260 + 156 = 4652 \text{ грн.}$$

Звідси загальні витрати на оплату праці ($B_{о.п.}$) визначаються за формулою:

$$В_{о.п.} = З_{осн.} + З_{дод.} \quad (4.3)$$

$$В_{о.п} = 46\,520 + 4652 = 51\,172 \text{ грн.}$$

Єдиний соціальний внесок (ЄСВ – 22%) визначається за формулою:

$$В_{єсв} = Воп * 0,22 \quad (4.4)$$

$$В_{єсв} = 51\,172 * 0,22 = 11\,258 \text{ грн.}$$

Проведені розрахунки витрат на оплату праці наведено у таблиці 4.2.

Таблиця 4.2 – Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівників	Основна заробітна плата, грн.			Додаткова заробітна плата, грн.	ЄСВ, грн.	Всього витрати на оплату праці, грн. 6 = 3+4+5
		Тарифна ставка, грн.	К-сть годин	Фактично нарах. зарплати, грн.			
		1	2	3	4	5	6
1	Кер. проєкту (Pm)	520	40	20 800	2080	5033	27 913
2	Інженера (l2)	340	34	11 560	1156	2797	15 513
3	Інженера (l1)	150	84	12 600	1260	3049	16 909
4	Тестувальник	130	12	1560	156	377	2093
Разом				46 520	4 652	11 258	62 430

Отже, загальні витрати на оплату праці становлять 62430 грн.

4.3. Розрахунок витрат на електроенергію

Для забезпечення життєдіяльності технічних засобів під час виконання робіт розраховується вартість використаної електроенергії. Витрати на живлення одного робочого місця визначаються наступним чином:

$$З_в = W * Т * S \quad (4.5)$$

де: W – необхідна потужність, кВт; Т – кількість годин роботи обладнання;
S – вартість кіловат-години електроенергії (приймаємо 15, 94 грн).

В нашій системі є 1 ПК. Витрати на електроенергію для цього комп'ютера обчислимо окремо, взявши за основу, що час роботи обладнання обчислюється в залежності від виконуваних робіт (згідно табл. 4.1) і споживані потужності наступні: комп'ютер – 0,82 кВт/год.

$$З_{ек} = 0,82*170*15,94 = 2222 \text{ грн.}$$

Таким чином, сума витрат на енергозабезпечення становить 2 222 грн.

4.4. Розрахунок суми амортизаційних відрахувань вебзастосунку «Burunduk Garage»

Функціонування та матеріальне зношування основних засобів у процесі створення інформаційного продукту потребує врахування механізмів їх поступового відновлення у грошовій формі через амортизацію.

Комп'ютерна техніка за класифікацією відноситься до 4-ї групи основних фондів. Розрахунок норми амортизації базується на припущенні, що мінімальний період корисного використання технічних засобів складає 2 роки. Величина амортизаційного фонду за час розробки розраховується за формулою:

$$А = \frac{Б_В * Н_А}{100\%} * T, \quad (4.6)$$

де: А – амортизаційні відрахування за звітний період, грн.;
$Б_В$ – балансова вартість групи основних фондів на початок звітного періоду, грн.;
$Н_А$ – норма амортизації, 0,04 %.

Враховуючи, що для написання коду вебплатформи «Burunduk Garage» та його верифікації використовувався один ПК вартістю 50 000,00 грн, обсяг

амортиціації дорівнює:

$$A = \frac{50\,000{,}00 * 0{,}04}{150} * 170 = 2267 \text{ грн.}$$

4.5. Обчислення накладних витрат

Накладні витрати включають загальногосподарські видатки: адміністрування, експлуатаційне утримання приміщень, організацію належних умов праці та інші супутні управлінські процеси. Залежно від структури підприємства вони варіюються в межах 20–60% від загального обсягу витрат на оплату праці.

$$Н_в = В_{о.п.} * \quad 0{,}2..0{,}6 \qquad (4.7)$$

де: $Н_В$ – накладні витрати.

$$Н_В = 51\,172 * 0{,}4 = 20\,469 \text{ грн.}$$

4.6. Складання кошторису витрат та визначення собівартості вебзастосунку «Burunduk Garage»

З метою формування підсумкової собівартості розробки вебзастосунку «Burunduk Garage», усі раніше розраховані статті витрат об'єднуються в єдину калькуляційну структуру (табл. 4.4).

Таблиця 4.4 - Структура кошторису та собівартості проєкту «Burunduk Garage»

№	Зміст витрат	Сума, грн.	В % до загальної суми
1.	Витрати на оплату праці	62 430	71,5
2.	Витрати на електроенергію	2 222	2,5
3.	Амортизаційні відрахування	2267	2,6
4.	Накладні витрати	20 469	23,4
5.	Собівартість	87 388	118

Собівартість ($С_В$) НДР розрахуємо за формулою:

$$Св = Во.п + Вс.з + Зе + А + Нв \quad (4.8)$$

Отже, собівартість дорівнює $С_В$= 87 388 грн.

4.7. Розрахунок ціни вебзастосунку «Burunduk Garage»

Визначення кінцевої ринкової вартості розробленого програмного продукту базується на врахуванні його собівартості, закладеного нормативу рентабельності бізнесу, а також обов'язкових податкових зборів.

Ціну робіт можна визначити за формулою:

$$Ц = Св * (1+Р_{рен}) * (1+ПДВ), \quad (4.9)$$

де: Св – собівартість; $Р_{рен.}$ – рівень рентабельності; ПДВ – ставка податку на додану вартість.

$$Ц = 87\ 388*(1+0,3)*(1+0,2)= 136\ 325 \text{ грн.}$$

4.8. Визначення економічної ефективності і терміну окупності капітальних вкладень

Оцінка результативності впровадження системи відображає ступінь ефективності використання залучених фінансових та людських ресурсів компанії. Для підтвердження комерційної доцільності вебплатформи «Burunduk Garage» розраховується показник чистої теперішньої вартості (ЧТВ) та термін повернення інвестицій (Ток)

$$ЧТВ = -Св + \sum_{i=1}^{t} \frac{Г_П}{(1+i)^t}, \quad (4.10)$$

де: Св – собівартість розробки; $Г_п$ – грошовий потік за t – ий рік; t – відповідний рік проекту; і – величина дисконтної ставки (10…15%).

$$ЧТВ = -87\ 388 + \frac{48\ 937}{(1+0,1)^1} + \frac{48\ 937}{(1+0,1)^2} + \frac{48\ 937}{(1+0,1)^3} = 34\ 337 \text{ грн}$$

Оскільки результуючий показник ЧТВ ≥ 0, даний інвестиційний проєкт є фінансово спроможним і рекомендується до практичної реалізації.

Термін окупності визначається за формулою:

$$T_{ОК} = T_{ПВ} + \frac{Н_В}{Г_{ПР}} \quad 4.11)$$

де: $Т_{ПВ}$ – період до повного відшкодування витрат, років; $Н_В$ – невідшкодовані витрати на початок року, грн.; $Г_{ПР}$ – грошовий потік на початок року, грн.

Розрахунок термін окупності:

$$Т_{ок} = 2 + \frac{2\,456}{48\,937} = 2{,}1 \text{ р.}$$

Таблиця 4.5 – Техніко-економічні показники «Burunduk Garage»

№ п/п	Показник	Значення
1.	Собівартість, грн.	87 388
2.	Плановий прибуток або грошовий потік, грн.	48 937
3.	Ціна, грн.	136 325
4.	Чиста теперішня вартість, грн.	34 377
5.	Термін окупності, рік	2,1

Позитивний рівень чистої теперішньої вартості та відносно короткий термін повернення витрат підтверджують високу інвестиційну привабливість і фінансову стабільність вебзастосунку. На основі розрахованих даних можна зробити остаточний висновок, що розробка автоматизованої системи «Burunduk Garage» є цілком доцільною, обґрунтованою та економічно вигідною.

5 ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ

5.1 Несприятливі виробничі фактори

У сучасних умовах промислового виробництва та стрімкого розвитку технологічних процесів безпека праці залишається одним із найголовніших пріоритетів будь-якого підприємства. Кожне робоче місце тією чи іншою мірою характеризується наявністю специфічних умов, які за певних обставин можуть трансформуватися у загрозу для здоров'я людини. Несприятливі виробничі фактори є складним комплексом чинників матеріального середовища та трудового процесу, які чинять негативний вплив на фізіологічний і психологічний стан робітника, знижують його працездатність та можуть призвести до розвитку професійних захворювань чи травматизму.

Традиційна наукова класифікація розподіляє ці чинники на чотири великі групи, кожна з яких має унікальну природу походження та специфіку дії. Першу та найбільш масштабну групу становлять фізичні фактори. Сюди відносять аномальні параметри мікроклімату, такі як надмірно висока або низька температура повітря, підвищена вологість та швидкість руху повітряних мас. Тривале перебування в умовах термічного стресу порушує процеси терморегуляції організму, викликаючи або перегрівання, або переохолодження, що негативно позначається на функціях серцево-судинної та нервової систем.

Не менш небезпечним є акустичне навантаження, що виражається у формі виробничого шуму, ультразвуку та інфразвуку. Постійне шумове забруднення, що перевищує допустимі санітарні норми, призводить не лише до специфічного ураження слухового апарату у вигляді професійної приглухуватості, а й викликає системні розлади: підвищення артеріального тиску, хронічну втому та неврози. Поряд із шумом часто виступає виробнича вібрація, яка за характером дії поділяється на загальну та локальну. Тривалий вплив локальної вібрації від ручного механізованого інструменту спричиняє розвиток небезпечної вібраційної хвороби, що характеризується спазмами судин, порушенням трофіки тканин та

стійким болем у кінцівках.

До фізичних факторів також зараховують незадовільні умови освітленості робочих зон. Недостатня кількість світла, наявність блiковості, різкі тіні або надмірна пульсація світлового потоку змушують працівника постійно напружувати органи зору. Це викликає швидку зорову втому, головний біль і суттєво підвищує ризик виникнення помилкових дій, які ведуть до аварійних ситуацій. Окреме місце посідають іонізуючі та неіонізуючі випромінювання, включаючи електромагнітні поля різних частотних діапазонів, які здатні накопичувати свій негативний ефект у тканинах організму, викликаючи глибокі патологічні зміни на клітинному рівні.

Другу групу формують хімічні фактори, які представлені різноманітними газами, парами, аерозолями, рідинами та пилом. Хімічні речовини можуть проникати в організм через дихальні шляхи, шлунково-кишковий тракт або безпосередньо через шкірні покриви. За характером патологічного впливу на людину їх класифікують на токсичні, подразнюючі, сенсибілізуючі, канцерогенні, мутагенні та такі, що впливають на репродуктивну функцію. Особливу підступність мають речовини з кумулятивним ефектом, які здатні роками накопичуватися в органах-депо (кістках, печінці, нирках) і проявлятися у вигляді важких хронічних отруєнь лише через значний проміжок часу. Виробничий пил, як окремий вид дисперсних систем, викликає фіброгенні процеси в легенях, результатом яких стають такі невиліковні захворювання, як пневмоконіози та силікози.

Біологічні фактори складають третю групу несприятливих чинників і включають патогенні мікроорганізми, серед яких бактерії, віруси, грибки, рикетсії, а також продукти їхньої життєдіяльності. Найбільшого впливу цієї групи зазнають працівники медичної сфери, мікробіологічної промисловості, сільського господарства та підприємств з очищення стічних вод. Взаємодія з біологічними агентами загрожує виникненням інфекційних захворювань, мікозів та важких алергічних реакцій, що важко піддаються стандартній терапії.

Четверта група охоплює фактори трудового процесу, які поділяють на

важкість та напруженість праці. Важкість праці відображає рівень фізичного навантаження на опорно-руховий апарат, серцево-судинну та дихальну системи. Вона визначається масою вантажу, що піднімається, величиною статичного та динамічного зусиль, формою робочої пози та робочим кроком. Напруженість праці характеризує емоційне та розумове навантаження, що пов'язане з переробкою великих обсягів інформації, дефіцитом часу, високою відповідальністю за результат та монотонністю праці. Постійна психоемоційна напруга призводить до синдрому професійного вигорання і нервового виснаження.

Мінімізація та повне усунення негативного впливу цих чотирьох груп факторів вимагає впровадження комплексного підходу. Він передбачає модернізацію обладнання, автоматизацію шкідливих ділянок, організацію раціонального режиму праці та відпочинку, а також безперервний моніторинг стану виробничого середовища. Лише через систематичний контроль і впровадження інженерно-технічних інновацій можна забезпечити формування безпечного простору, де ризики для здоров'я персоналу зведені до мінімуму.

5.2 Захисні засоби з електробезпеки

Забезпечення надійного захисту персоналу під час експлуатації та обслуговування електроустановок є одним із найбільш жорстко регламентованих напрямків у системі охорони праці. Електричний струм має унікальну та вкрай небезпечну властивість - він не має видимих ознак загрози, його неможливо виявити на відстані за допомогою органів чуття, а його дія на живий організм є блискавичною та руйнівною. Саме тому використання спеціалізованих захисних засобів є обов'язковою та безальтернативною умовою для запобігання ураженням електричним струмом, виникненню електричної дуги чи впливу електромагнітних полів.

Усі електрозахисні засоби, що застосовуються на виробництві, проходять сувору класифікацію та поділ залежно від робочої напруги обладнання та призначення самого засобу. Загальноприйнятим є поділ на ізолювальні,

огороджувальні та запобіжні засоби, причому ізолювальні відіграють ключову роль безпосередньо при виконанні маніпуляцій під напругою. Вони, у свою чергу, поділяються на дві фундаментальні категорії: основні та додаткові електрозахисні засоби.

До категорії основних засобів в електроустановках напругою до 1000 В відносять ті пристрої, ізоляція яких здатна тривалий час надійно витримувати робочу напругу мережі. Це означає, що працівник може безпечно торкатися струмоведучих частин, які перебувають під напругою, використовуючи виключно цей засіб. Сюди належать ізолювальні штанги всіх типів, ізолювальні та електровимірювальні кліщі, покажчики напруги, а також діелектричні рукавички та слюсарно-монтажний інструмент з ізолювальними рукоятками. Якщо ж мова йде про установки з напругою понад 1000 В, то перелік основних засобів звужується до ізолювальних штанг, кліщів та складних покажчиків напруги, оскільки високий потенціал вимагає значно більшої товщини та специфіки ізоляційного шару.

Додаткові електрозахисні засоби мають зовсім інше функціональне призначення. Самі по собі вони не здатні забезпечити стовідсотковий захист від ураження струмом при робочій напрузі, тому їх категорично заборонено використовувати самостійно без основних засобів. Їхнє головне завдання — посилити захисний ефект основних засобів, а також убезпечити робітника від таких специфічних явищ, як напруга кроку та напруга дотику. В установках до 1000 В як додаткові засоби використовують діелектричне взуття (калоші), діелектричні килимки та ізолювальні підставки. В електромережах понад 1000 В цей перелік доповнюється діелектричними ботами, які мають вищий рівень електричної міцності порівняно з калошами, та спеціальними ізолювальними накладками й ковпаками.

Окрему важливу групу становлять огороджувальні та заземлювальні захисні засоби. Тимчасові переносні заземлення є найвищим ступенем захисту при виконанні робіт на відключених струмоведучих частинах. Вони призначені на випадок помилкової подачі напруги на робоче місце або появи наведеної напруги

від сусідніх ліній. У такій ситуації переносне заземлення викликає миттєве коротке замикання, що призводить до спрацьовування систем автоматичного захисту (запобіжників чи вимикачів) та повного знеструмлення ділянки. До цієї ж категорії відносять переносні огородження, ізолювальні накладки, а також плакати та знаки безпеки, які виконують інформаційну та попереджувальну функції, забороняючи вмикання апаратів або вказуючи на межі безпечної зони.

Окрім електрозахисних засобів, персонал обов'язково забезпечується засобами індивідуального захисту загального призначення, які адаптовані під специфіку енергетики. Сюди входять захисні каски для запобігання механічним травмам голови, захисні окуляри та щитки, що бережуть очі від сліпучого спалаху та бризок розплавленого металу при виникненні електричної дуги, а також спеціальний одяг із термостійких матеріалів.

Ефективність та надійність усіх захисних засобів залежить не лише від їхньої конструкції, а й від дотримання суворих правил експлуатації, зберігання та періодичних випробувань. Кожен засіб захисту повинен регулярно проходити ретельний візуальний огляд на предмет відсутності тріщин, проколів, розшарувань чи забруднень. Крім того, нормативними документами встановлені жорсткі терміни проведення періодичних електричних та механічних лабораторних випробувань під підвищеною напругою. На кожному засобі, що пройшов перевірку, обов'язково проставляється штамп із зазначенням максимально допустимої напруги та дати наступного випробування, що гарантує безпеку життя працівника в екстремальних умовах виробництва.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено вебзастосунок для автоматизації управління автопарком «Burunduk Garage». Система реалізує публічний каталог автомобілів, сторінку деталей, оформлення бронювання, особистий кабінет користувача, адміністративне керування автомобілями, користувачами, бронюваннями і статистикою.

Під час роботи було проаналізовано предметну область автопрокату, визначено проблеми ручного обліку та сформовано технічне завдання. На основі вимог спроєктовано структуру даних, API, ролі доступу і користувацькі сценарії.

Клієнтська частина розроблена з використанням React, Vite, React Router і Tailwind CSS. Серверна частина реалізована на Node.js та Express із використанням MongoDB і Mongoose. Для авторизації використано JWT, для захисту паролів - bcrypt, для роботи із зображеннями - ImageKit.

Особливу увагу приділено практичним можливостям системи: пошуку і фільтрації бронювань, редагуванню додаткових опцій, обмеженню редагування оренди для користувача за два дні до початку, адаптивності, коректному сортуванню, відображенню електромобілів і редактору позиції фотографій автомобілів.

Розроблений вебзастосунок може бути використаний як основа для подальшого впровадження у сервісі оренди автомобілів. У майбутньому систему можна розширити онлайн-оплатою, календарем зайнятості, сповіщеннями, інтеграціями та детальнішою аналітикою.

ПЕРЕЛІК ПОСИЛАНЬ

1. React Documentation. URL: https://react.dev/
2. Vite Documentation. URL: https://vitejs.dev/
3. Express Documentation. URL: https://expressjs.com/
4. Mongoose Documentation. URL: https://mongoosejs.com/
5. MongoDB Manual. URL: https://www.mongodb.com/docs/
6. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs
7. JSON Web Tokens. URL: https://jwt.io/
8. ImageKit Documentation. URL: https://docs.imagekit.io/
9. MDN Web Docs. JavaScript, HTML, CSS references. URL: https://developer.mozilla.org/
10. Методичні вказівки до виконання кваліфікаційної роботи зі спеціальності «Комп'ютерні науки».
11. Бондаренко М.Ф., Білоус Н.В. Вебтехнології та вебдизайн. Київ : Видавнича група BHV, 2021. 384 с. https://media.chnu.edu.ua/media/0mnpydhj/122-compscience.pdf?utm_source=chatgpt.com
12. Vercel Documentation. URL: https://vercel.com/docs
13. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/
14. Морзе Н.В., Вембер В.П., Кузьмінська О.Г. Основи інформаційних технологій. Київ : Академвидав, 2020. 352 с. https://scholar.google.com/citations?hl=ru&user=b18TCTYAAAAJ&utm_source=chatgpt.com
15. Завадський І.О. Основи баз даних. Київ : Видавнича група BHV, 2019. 288 с. https://programming.in.ua/other-files/other/197-book-basis-database?utm_source=chatgpt.com

ДОДАТКИ

Додаток А. Лістинг моделей бази даних

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true, match: /^[\w-
\.]+@gmail\.com$/ },
  password: { type: String, required: true },
  phone_number: { type: String, required: true },
  first_name: { type: String, required: true },
  last_name: { type: String, required: true },
  middle_name: { type: String, required: true },
  role: { type: String, required: true, enum: ['admin', 'user'] },
  created_at: { type: Date, default: Date.now }
});
const bodyTypeSchema = new mongoose.Schema({
  type_name: { type: String, required: true, unique: true }
});
const classSchema = new mongoose.Schema({
  class_name: { type: String, required: true, unique: true }
});
const fuelTypeSchema = new mongoose.Schema({
  fuel_type: { type: String, required: true, unique: true }
});
const statusSchema = new mongoose.Schema({
  status: { type: Boolean, required: true, unique: true }
});
const rentalOptionItemSchema = new mongoose.Schema(
  {
    enabled: { type: Boolean, default: false },
    km: { type: Number, default: 0 },
    price: { type: Number, default: 0 },
  },
```

```
  { _id: false }
);
const rentalFlatOptionSchema = new mongoose.Schema(
     { enabled: { type: Boolean, default: false },
    price: { type: Number, default: 0 },   },
  { _id: false } );
const rentalSchema = new mongoose.Schema({
  client_id: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required:
true },
  car_id: { type: mongoose.Schema.Types.ObjectId, ref: 'Car', required: true
},
  start_date: { type: Date, required: true },
  end_date: { type: Date, required: true },
  base_rental_price: { type: Number, default: 0 },
  options: {
    delivery: { type: rentalOptionItemSchema, default: () => ({}) },
    return_elsewhere: { type: rentalOptionItemSchema, default: () => ({}) },
    child_seat: { type: rentalFlatOptionSchema, default: () => ({}) },
    bike_rack: { type: rentalFlatOptionSchema, default: () => ({}) },
    full_insurance: { type: rentalFlatOptionSchema, default: () => ({}) },
  },
  status: { type: String, enum: ['active', 'cancelled'], default: 'active'
},
  total_price: { type: Number, required: true },
  created_at: { type: Date, default: Date.now } });
const reviewSchema = new mongoose.Schema({
  client: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required:
true },
  car: { type: mongoose.Schema.Types.ObjectId, ref: 'Car', required: true },
  car_name: { type: String },
  comment: { type: String },
  review_date: { type: Date }
});
const carSchema = new mongoose.Schema({
```

```
  name: { type: String, required: true },
  body_type: { type: mongoose.Schema.Types.ObjectId, ref: 'BodyType' },
  class: { type: mongoose.Schema.Types.ObjectId, ref: 'Class' },
  engine_volume: { type: Number },
  horsepower: { type: Number },
  fuel_type: { type: mongoose.Schema.Types.ObjectId, ref: 'FuelType' },
  fuel_consumption: { type: String },
  color: { type: String, required: true },
  price_per_day: { type: Number, required: true },
  status: { type: mongoose.Schema.Types.ObjectId, ref: 'Status', default:
null },
  photo: { type: String },
  image_position_x: { type: Number, default: 0 },
  image_position_y: { type: Number, default: 0 },
  image_zoom: { type: Number, default: 1 },
  last_modified: { type: Date, default: Date.now } });
const User = mongoose.model('User', userSchema);
const BodyType = mongoose.model('BodyType', bodyTypeSchema);
const Class = mongoose.model('Class', classSchema);
const FuelType = mongoose.model('FuelType', fuelTypeSchema);
const Status = mongoose.model('Status', statusSchema);
const Car = mongoose.model('Car', carSchema);
const Rental = mongoose.model('Rental', rentalSchema);
const Review = mongoose.model('Review', reviewSchema);
module.exports = {
  User,
  BodyType,
  Class,
  FuelType,
  Status,
  Car,
  Rental,
  Review
};
```

Додаток Б. Лістинг серверної частини Express

```
const express = require('express');
const cors = require('cors');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
const { body, validationResult } = require('express-validator');
const connectDB = require('./config/database');
const { User, BodyType, Class, FuelType, Status, Car, Review, Rental } =
require('./models');
const mongoose = require('mongoose');
const { uploadImageFromUrl } = require('./utils/imageKit');
const {
  applyRentalTimes,
  applyRentalStartTime,
  applyRentalEndTime,
  validateRentalPrice,
  calculateRentalPrice,
  normalizeOptions,
} = require('./utils/rentalPricing');
const app = express();
const port = process.env.PORT || 3001;
const RENTAL_EDIT_DEADLINE_MS = 2 * 24 * 60 * 60 * 1000;
// Middleware
app.use(cors({
  origin: [
    'https://burundukgarage.vercel.app',
    'https://kursova3-2.onrender.com',
    'http://localhost:5173', // For local development
    'http://localhost:3000'  // For local development
  ],
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization'],
  credentials: true
```

```
}));
app.use(express.json());
connectDB();
const auth = (req, res, next) => {
  const token = req.header('Authorization')?.replace('Bearer ', '');

  if (!token) {
    return res.status(401).json({ error: 'Токен не надано' });
  }
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET ||
'BurundukGarage');
    req.user = decoded;
    next();
  } catch (error) {
    res.status(401).json({ error: 'Недійсний токен' });
  }
};
const isAdmin = (req, res, next) => {
  if (req.user.role !== 'admin') {
    return res.status(403).json({ error: 'Потрібні права адміністратора' });
  }
  next();
};
const canModifyBeforeRental = (startDate) =>
  new Date(startDate).getTime() - Date.now() >= RENTAL_EDIT_DEADLINE_MS;
const toFiniteNumber = (value, fallback) => {
  const number = Number(value);
  return Number.isFinite(number) ? number : fallback;
};
const buildOptionsSnapshot = (normalized, pricing) => ({
  delivery: {
    enabled: normalized.delivery,
    km: normalized.delivery ? normalized.deliveryKm : 0,
```

```
    price: pricing.deliveryPrice,
  },
  return_elsewhere: {
    enabled: normalized.returnElsewhere,
    km: normalized.returnElsewhere ? normalized.returnKm : 0,
    price: pricing.returnPrice,
  },
  child_seat: {
    enabled: normalized.childSeat,
    price: pricing.childSeatPrice,
  },
  bike_rack: {
    enabled: normalized.bikeRack,
    price: pricing.bikeRackPrice,
  },
  full_insurance: {
    enabled: normalized.fullInsurance,
    price: pricing.insurancePrice,
  },
});
const validateRentalOptions = (normalized) => {
  if (normalized.delivery && normalized.deliveryKm <= 0) {
    return 'Вкажіть відстань для доставки авто';
  }
  if (normalized.returnElsewhere && normalized.returnKm <= 0) {
    return 'Вкажіть відстань для повернення в іншому місці';
  }
  return null;
};
app.get('/api/body-types', async (req, res) => {
  try {
    const bodyTypes = await BodyType.find();
    res.json(bodyTypes);
  } catch (error) {
```

```
    console.error('Помилка отримання типів кузова:', error);
    res.status(500).json({ error: 'Не вдалося отримати типи кузова' });
  }
});
app.get('/api/classes', async (req, res) => {
  try {
    const classes = await Class.find();
    res.json(classes);
  } catch (error) {
    console.error('Помилка отримання класів:', error);
    res.status(500).json({ error: 'Не вдалося отримати класи' });
  }
});
app.get('/api/fuel-types', async (req, res) => {
  try {
    const fuelTypes = await FuelType.find();
    res.json(fuelTypes);
  } catch (error) {
    console.error('Помилка отримання типів палива:', error);
    res.status(500).json({ error: 'Не вдалося отримати типи палива' });
  }
});
app.get('/api/statuses', async (req, res) => {
  try {
    const statuses = await Status.find();
    res.json(statuses);
  } catch (error) {
    console.error('Помилка отримання статусів:', error);
    res.status(500).json({ error: 'Не вдалося отримати статуси' });
  }
});
app.get('/api/cars', async (req, res) => {
// ... фрагмент скорочено для пояснювальної записки ...
```

Додаток В. Лістинг модуля ImageKit

```
const ImageKit = require('imagekit');
const axios = require('axios');
require('dotenv').config();
const imagekit = new ImageKit({
  publicKey: process.env.IMAGEKIT_PUBLIC_KEY,
  privateKey: process.env.IMAGEKIT_PRIVATE_KEY,
  urlEndpoint: process.env.IMAGEKIT_URL_ENDPOINT
});
const BG_REMOVE_TRANSFORMATION = 'tr:e-bgremove';
const withBackgroundRemoval = (imageUrl) => {
  if (!imageUrl || imageUrl.includes(BG_REMOVE_TRANSFORMATION)) {
    return imageUrl;
  }
  const [baseUrl, queryString] = imageUrl.split('?');
  const endpoint = (process.env.IMAGEKIT_URL_ENDPOINT || '').replace(/\/$/,
'');
  if (endpoint && baseUrl.startsWith(endpoint)) {
    const imagePath = baseUrl.slice(endpoint.length).replace(/^\//, '');
    return
`${endpoint}/${BG_REMOVE_TRANSFORMATION}/${imagePath}${queryString        ?
`?${queryString}` : ''}`;
  }
  const marker = 'ik.imagekit.io/';
  const markerIndex = baseUrl.indexOf(marker);
  if (markerIndex !== -1) {
    const pathStart = baseUrl.indexOf('/', markerIndex + marker.length);
    if (pathStart !== -1) {
      return                                                  `${baseUrl.slice(0,
pathStart)}/${BG_REMOVE_TRANSFORMATION}${baseUrl.slice(pathStart)}${querySt
ring ? `?${queryString
    }
  }
```

```
  return imageUrl; };
/**
 * Upload image from URL to ImageKit
 * @param {string} imageUrl - URL of the image to upload
 * @param {string} fileName - Name for the uploaded file
 *  @returns {Promise<string>} - ImageKit URL with background removal
transformation  */
const uploadImageFromUrl = async (imageUrl, fileName) => {
  try {
    // If the URL is already an ImageKit URL, save the transformed URL
directly.
    if (imageUrl && imageUrl.includes('ik.imagekit.io')) {
      return withBackgroundRemoval(imageUrl);
    }
    // Fetch the image from the URL
    const response = await axios.get(imageUrl, {
      responseType: 'arraybuffer'
    });
    const buffer = Buffer.from(response.data);
    // Upload to ImageKit
    const result = await imagekit.upload({
      file: buffer,
      fileName: fileName,
      useUniqueFileName: true
    });
    return withBackgroundRemoval(result.url);
  } catch (error) {
    console.error('Error uploading image to ImageKit:', error);
    throw error;
  }
};
module.exports = {
  uploadImageFromUrl,
  withBackgroundRemoval};
```

Додаток Г. Лістинг сервісу API

```
const API_URL = import.meta.env.VITE_API_URL || 'http://localhost:3001/api';
class ApiService {
    static getToken() {
        return localStorage.getItem('token');
    }
    static clearToken() {
        localStorage.removeItem('token');
    }
    static isTokenExpired(token) {
        if (!token) return true;
        try {
            const base64Url = token.split('.')[1];
            const base64 = base64Url.replace(/-/g, '+').replace(/_/g, '/');
            const                          jsonPayload                          =
decodeURIComponent(atob(base64).split('').map(c => {
                return '%' + ('00' + c.charCodeAt(0).toString(16)).slice(-
2);
            }).join(''));
            const { exp } = JSON.parse(jsonPayload);
            return Date.now() >= exp * 1000;
        } catch (error) {
            console.error('Token validation error:', error);
            return true;
        }
    }
    static async request(endpoint, options = {}) {
        const url = `${API_URL}${endpoint}`;
        const headers = {
            'Content-Type': 'application/json',
            ...options.headers
        };
        const token = this.getToken();
```

```
        if (token) {
            if (this.isTokenExpired(token)) {
                console.log('Термін дії токена закінчився, очищення токена');
                this.clearToken();
                if (!endpoint.startsWith('/cars/') || options.method !== 'GET') {
                    window.location.href = '/login';
                }
                throw new Error('Термін дії токена закінчився');
            }
            headers['Authorization'] = `Bearer ${token}`;
        }
        try {
            const response = await fetch(url, {
                ...options,
                headers
            });
            if (!response.ok) {
                const error = await response.json().catch(() => ({ error: 'Виникла невідома помилка' }));
                if (response.status === 401) {
                    this.clearToken();
                    if (!endpoint.startsWith('/cars/') || options.method !== 'GET') {
                        window.location.href = '/login';
                    }
                }
                throw new Error(error.error || error.message || 'Помилка запиту');
            }
            const data = await response.json();
            return data;
        } catch (error) {
```

```
            console.error('Помилка API запиту:', error);
            throw error;
        }
    }
    // Auth endpoints
    static async login(email, password) {
        return this.request('/login', {
            method: 'POST',
            body: JSON.stringify({ email, password })
        });
    }
    static async signup(userData) {
        return this.request('/signup', {
            method: 'POST',
            body: JSON.stringify(userData)
        });
    }
    // Reference data endpoints
    static async getReferenceData() {
        const [bodyTypes, classes, fuelTypes, statuses] = await
Promise.all([
            this.request('/body-types'),
            this.request('/classes'),
            this.request('/fuel-types'),
            this.request('/statuses')
        ]);
        return { bodyTypes, classes, fuelTypes, statuses };
    }
    // Cars endpoints
    static async getCars(params = {}) {
        const queryString = new URLSearchParams(params).toString();
        return this.request(`/cars${queryString ? `?${queryString}` : ''}`);
    }
```

```
    static async getCar(id) {
        return this.request(`/cars/${id}`);
    }
    static async getCarReviews(id) {
        return this.request(`/cars/${id}/reviews`);
    }
    static async createReview(carId, reviewData) {
        return this.request(`/cars/${carId}/reviews`, {
            method: 'POST',
            body: JSON.stringify(reviewData)
        });
    }
    static async createCar(carData) {
        return this.request('/cars', {
            method: 'POST',
            body: JSON.stringify(carData)
        });
    }
    static async updateCar(id, carData) {
        return this.request(`/cars/${id}`, {
            method: 'PUT',
            body: JSON.stringify(carData)
        });
    }
    static async deleteCar(id) {
        return this.request(`/cars/${id}`, {
            method: 'DELETE'
        });
    }
// ... фрагмент скорочено для пояснювальної записки ...
```

Додаток Д. Лістинг компонента автопарку

```
import { useState, useEffect, useCallback } from 'react';
import { useAuth } from '../context/AuthContext';
import { Link, useSearchParams } from 'react-router-dom';
import ApiService from '../services/api';
import CarForm from './CarForm';
import FilterForm from './FilterForm';
import {
    formatCardCapacity,
    formatCardConsumption,
    getCarImageStyle,
    getCarSpecLabels,
    withImageKitBackgroundRemoval,
} from '../utils/carDisplay';
const formatCardEngineVolume = (value, car) => {
    const labels = getCarSpecLabels(car);
    if (labels.electric) return formatCardCapacity(value, car);

    const match = String(value ?? '').replace(',',
'.').match(/\d+(\.\d+)?/);
    const number = Number(match?.[0]);
    return Number.isFinite(number) ? number.toFixed(1) :
formatCardCapacity(value, car);
};

const CarList = () => {
    const [searchParams, setSearchParams] = useSearchParams();
    const [cars, setCars] = useState([]);
    const [loading, setLoading] = useState(false);
    const [error, setError] = useState(null);
    const [searchTerm, setSearchTerm] = useState('');
    const [filter, setFilter] = useState('');
    const [page, setPage] = useState(parseInt(searchParams.get('page') ||
```

```
'1'));
    const [sortBy, setSortBy] = useState('name');
    const [order, setOrder] = useState('ASC');
    const { user } = useAuth();
    const [editingCar, setEditingCar] = useState(null);
    const [isEditModalOpen, setIsEditModalOpen] = useState(false);
    const [isFilterModalOpen, setIsFilterModalOpen] = useState(false);
    const [activeFilters, setActiveFilters] = useState({});



    useEffect(() => {
        const handleCarListUpdate = () => {
            console.log('Car list update event received');
            fetchCars();
        };
    useEffect(() => {
        const timer = setTimeout(() => {
            setFilter(searchTerm);
        }, 1000);

        return () => clearTimeout(timer);
    }, [searchTerm]);

    useEffect(() => {
        setSearchParams({ page: page.toString() });
    }, [page, setSearchParams]);

    const formatFilterValue = (key, value) => {
        if (key.endsWith('Name')) return null;
        const  specLabels  =  getCarSpecLabels(activeFilters.fuelTypeName  ||
'');
        const displayKey = {
            minPrice: 'Ціна від',
            maxPrice: 'Ціна до',
```

```
            minEngineVolume: `${specLabels.capacityLabel} від`,
            maxEngineVolume: `${specLabels.capacityLabel} до`,
            minHorsepower: 'Потужність від',
            maxHorsepower: 'Потужність до',
            bodyType: 'Тип кузова',
            class: 'Клас',
            fuelType: 'Тип палива',
            color: 'Колір',
            available: 'Доступність'
        }[key];
        let displayValue = value;
        if (key === 'bodyType') {
            displayValue = activeFilters.bodyTypeName || value;
        } else if (key === 'class') {
            displayValue = activeFilters.className || value;
        } else if (key === 'fuelType') {
            displayValue = activeFilters.fuelTypeName || value;
        } else if (key === 'available') {
            displayValue = value === 'true' ? 'Доступний' : 'Недоступний';
        } else if (key.startsWith('min') || key.startsWith('max')) {
            if (key.includes('Price')) {
                displayValue = `${value}₴`;
            } else if (key.includes('EngineVolume')) {
                displayValue = `${value} ${specLabels.capacityUnit}`;
            } else if (key.includes('Horsepower')) {
                displayValue = `${value} к.с.`;
            }
        }

        return `${displayKey}: ${displayValue}`;
    };


// ... фрагмент скорочено для пояснювальної записки ...
```

Додаток Е. Лістинг форми автомобіля

```
import { useState, useEffect } from 'react';
import ApiService from '../services/api';
import { getCarImageStyle, getCarSpecLabels, withImageKitBackgroundRemoval
} from '../utils/carDisplay';

const CarForm = ({ onSubmit, initialData = null }) => {
    const [formData, setFormData] = useState({
        name: '',
        body_type_id: '',
        class_id: '',
        engine_volume: '',
        horsepower: '',
        fuel_type_id: '',
        fuel_consumption: '',
        color: '',
        price_per_day: '',
        status_id: '',
        photo: '',
        image_position_x: 0,
        image_position_y: 0,
        image_zoom: 1
    });
    const [referenceData, setReferenceData] = useState({
        bodyTypes: [],
        classes: [],
        fuelTypes: [],
        statuses: []
    });
    const [loading, setLoading] = useState(false);
    const [error, setError] = useState(null);

    const selectedFuelType = referenceData.fuelTypes.find(
```

```
        type => type._id === formData.fuel_type_id
    );
    const specLabels = getCarSpecLabels(selectedFuelType?.fuel_type ||
initialData);

    useEffect(() => {
        if (initialData) {
            setFormData({
                ...initialData,
                body_type_id:          initialData.body_type_id          ||
initialData.body_type?._id || '',
                class_id: initialData.class_id || initialData.class?._id ||
'',
                fuel_type_id:          initialData.fuel_type_id          ||
initialData.fuel_type?._id || '',
                status_id: initialData.status_id || initialData.status?._id
|| '',
                image_position_x: initialData.image_position_x ?? 0,
                image_position_y: initialData.image_position_y ?? 0,
                image_zoom: initialData.image_zoom ?? 1
            });
        }
    }, [initialData]);

    useEffect(() => {
        const fetchReferenceData = async () => {
            try {
                setLoading(true);
                const data = await ApiService.getReferenceData();
                setReferenceData(data);
            } catch (err) {
                setError(err.message);
            } finally {
                setLoading(false);
```

```
            }
        };

        fetchReferenceData();
    }, []);

    const handleChange = (e) => {
        const { name, value } = e.target;
        setFormData(prev => ({
            ...prev,
            [name]: value
        }));
    };

    const resetImagePosition = () => {
        setFormData(prev => ({
            ...prev,
            image_position_x: 0,
            image_position_y: 0,
            image_zoom: 1
        }));
    };

    const handleSubmit = async (e) => {
        e.preventDefault();
        try {
            setLoading(true);
            setError(null);

// ... фрагмент скорочено для пояснювальної записки ...
```

Додаток Ж. Лістинг адміністративної сторінки бронювань

```
import { useState, useEffect, useMemo, useCallback } from 'react';
import { Link } from 'react-router-dom';
import ApiService from '../services/api';
import RentalEditModal from './RentalEditModal';
import AppPageLayout, {
  pagePanelClass,
  pageTableHeadClass,
  pageTableCellClass,
  pageInputClass,
} from './AppPageLayout';
const formatDate = (dateString) => {
  try {
    if (!dateString) return 'Не вказано';
    const date = new Date(dateString);
    if (isNaN(date.getTime())) return 'Некоректна дата';
    return new Intl.DateTimeFormat('uk-UA', {
      day: '2-digit',
      month: '2-digit',
      year: 'numeric',
      hour: '2-digit',
      minute: '2-digit',
    }).format(date);
  } catch {
    return 'Помилка формату дати';
  }
};
const countOptions = (options) => {
  if (!options) return 0;
  return [
    options.delivery?.enabled,
    options.return_elsewhere?.enabled,
    options.child_seat?.enabled,
```

```
    options.bike_rack?.enabled,
    options.full_insurance?.enabled,
  ].filter(Boolean).length;
};
const getRentalStatus = (rental) => {
  if (rental.status === 'cancelled') {
    return {
      label: 'Скасовано',
      className: 'bg-red-500/15 text-red-300 border-red-500/25',
    };
  }
  const now = Date.now();
  const start = new Date(rental.start_date).getTime();
  const end = new Date(rental.end_date).getTime();
  if (now < start) {
    return {
      label: 'Майбутнє',
      className: 'bg-sky-500/15 text-sky-300 border-sky-500/25',
    };
  }
  if (now > end) {
    return {
      label: 'Завершено',
      className: 'bg-white/10 text-white/55 border-white/20',
    };
  }
  return {
    label: 'Активне',
    className: 'bg-emerald-500/15 text-emerald-300 border-emerald-500/25',
  };
};
const RentalsPage = () => {
  const [rentals, setRentals] = useState([]);
  const [loading, setLoading] = useState(true);
```

```
  const [error, setError] = useState(null);
  const [editingRental, setEditingRental] = useState(null);
  const [filters, setFilters] = useState({
    search: '',
    dateFrom: '',
    dateTo: '',
    status: '',
  });
  const [draftFilters, setDraftFilters] = useState({
    search: '',
    dateFrom: '',
    dateTo: '',
    status: '',
  });
  const fetchRentals = useCallback(async () => {
    try {
      setLoading(true);
      setError(null);
      const params = Object.fromEntries(
        Object.entries(filters).filter(([, value]) => value)
      );
      const data = await ApiService.getRentals(params);
      setRentals(data);
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  }, [filters]);

  useEffect(() => {
    fetchRentals();
  }, [fetchRentals]);
// ... фрагмент скорочено для пояснювальної записки ...
```

Додаток И. Лістинг сторінки «Мої оренди»

```
import { useState, useEffect, useMemo } from 'react';
import { Link } from 'react-router-dom';
import ApiService from '../services/api';
import RentalEditModal from './RentalEditModal';
import AppPageLayout, { pagePanelClass } from './AppPageLayout';

const RENTAL_EDIT_DEADLINE_MS = 2 * 24 * 60 * 60 * 1000;

const formatDate = (dateString) => {
  try {
    if (!dateString) return '—';
    const date = new Date(dateString);
    if (isNaN(date.getTime())) return '—';
    return new Intl.DateTimeFormat('uk-UA', {
      day: '2-digit',
      month: 'short',
      year: 'numeric',
      hour: '2-digit',
      minute: '2-digit',
    }).format(date);
  } catch {
    return '—';
  }
};

const getRentalStatusFromRental = (rental) => {
  if (rental.status === 'cancelled') {
    return {
      label: 'Скасована',
      badge: 'bg-red-500/20 text-red-300 border-red-500/30',
      accent: 'border-red-500',
      progress: 0,
```

```
    };
  }

  const now = Date.now();
  const start = new Date(rental.start_date).getTime();
  const end = new Date(rental.end_date).getTime();
  if (now < start) {
    return {
      label: 'Майбутня',
      badge: 'bg-sky-500/20 text-sky-300 border-sky-500/30',
      accent: 'border-sky-500',
      progress: 0,
    };
  }
  if (now > end) {
    return {
      label: 'Завершена',
      badge: 'bg-white/10 text-white/50 border-white/20',
      accent: 'border-white/20',
      progress: 100,
    };
  }
  const progress = Math.min(100, Math.max(0, ((now - start) / (end - start))
* 100));
  return {
    label: 'Активна',
    badge: 'bg-emerald-500/20 text-emerald-300 border-emerald-500/30',
    accent: 'border-emerald-500',
    progress: Math.round(progress),
  };
};

const canModifyRental = (rental) =>
  rental.status !== 'cancelled' &&
```

```
  new Date(rental.start_date).getTime() - Date.now() >=
RENTAL_EDIT_DEADLINE_MS;

const StatCard = ({ label, value, hint }) => (
  <div className={`${pagePanelClass} p-5`}>
    <p  className="text-xs  font-semibold  uppercase  tracking-wider  text-
white/45">{label}</p>
    <p className="mt-2 text-2xl font-bold">{value}</p>
    {hint && <p className="mt-1 text-sm text-white/50">{hint}</p>}
  </div>
);

const getSelectedOptions = (options) => {
  if (!options) return [];
  const items = [];
  if (options.delivery?.enabled) {
    items.push(`Доставка  додому  (${options.delivery.km}  км)  —
${options.delivery.price} грн`);
  }
  if (options.return_elsewhere?.enabled) {
    items.push(`Повернення в іншому місці (${options.return_elsewhere.km}
км) — ${options.return_elsewhere.price} грн`);
  }
  if (options.child_seat?.enabled) {
    items.push(`Дитяче крісло — ${options.child_seat.price} грн`);
  }
  if (options.bike_rack?.enabled) {
    items.push(`Кріплення  для  велосипедів  —  ${options.bike_rack.price}
грн`);
  }
  if (options.full_insurance?.enabled) {
    items.push(`Повне страхування — ${options.full_insurance.price} грн`);
  }
  return items;
```

```
};

const UserRentalsPage = () => {
  const [rentals, setRentals] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [editingRental, setEditingRental] = useState(null);

  useEffect(() => {
    fetchUserRentals();
  }, []);

  const fetchUserRentals = async () => {
    try {
      setLoading(true);
      setError(null);
      const userRentals = await ApiService.getRentals();
      setRentals(userRentals);
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };
// ... фрагмент скорочено для пояснювальної записки ...
```

Додаток К. Лістинг форматування характеристик автомобіля

```
export const isElectricCar = (carOrFuelType) => {
  const fuelType =
    typeof carOrFuelType === 'string'
      ? carOrFuelType
      : carOrFuelType?.fuel_type?.fuel_type || carOrFuelType?.fuel_type ||
'';
  return            fuelType.toLowerCase().includes('елект')            ||
fuelType.toLowerCase().includes('electric'); };
const firstNumber = (value) => {
  const match = String(value ?? '').replace(',', '.').match(/\d+(\.\d+)?/);
  return match ? match[0] : value; };
const fixedOneNumber = (value) => {
  const number = Number(firstNumber(value));
  return Number.isFinite(number) ? number.toFixed(1) : value;
};
export const getCarSpecLabels = (carOrFuelType) => {
  const electric = isElectricCar(carOrFuelType);
  return {
    electric,
    capacityLabel: electric ? 'Ємність батареї' : "Об'єм двигуна",
    capacityUnit: electric ? 'кВт·год' : 'л',
    consumptionLabel: electric ? 'Витрата енергії' : 'Розхід палива',
    consumptionUnit: electric ? 'кВт·год/100км' : 'л/100км',
  };
};
export const formatCapacity = (value, carOrFuelType) => {
  const labels = getCarSpecLabels(carOrFuelType);
  const   normalizedValue   =   labels.electric   ?   firstNumber(value)   :
fixedOneNumber(value);
  return `${normalizedValue} ${labels.capacityUnit}`;
};
export const formatConsumption = (value, carOrFuelType) => {
```

```
  const labels = getCarSpecLabels(carOrFuelType);
  return `${firstNumber(value)}${labels.consumptionUnit}`;
};
export const formatCardCapacity = (value, carOrFuelType) => {
  const labels = getCarSpecLabels(carOrFuelType);
  return labels.electric ? `${firstNumber(value)} кВт` : firstNumber(value);
};
export const formatCardConsumption = (value) => firstNumber(value);
export const getCarImageStyle = (car = {}) => {
  const x = Number(car.image_position_x) || 0;
  const y = Number(car.image_position_y) || 0;
  const zoomValue = Number(car.image_zoom);
  const zoom = Number.isFinite(zoomValue) && zoomValue > 0 ? zoomValue : 1;
  return {
    left: `calc(50% + ${x}%)`,
    top: `calc(50% + ${y}%)`,
    objectFit: 'contain',
    transform: `translate(-50%, -50%) scale(${zoom})`,
    transformOrigin: 'center center',
  };
};
export const withImageKitBackgroundRemoval = (imageUrl) => {
  if     (!imageUrl     ||     !imageUrl.includes('ik.imagekit.io')     ||
imageUrl.includes('tr:e-bgremove')) {
    return imageUrl;
  }
  const [baseUrl, queryString] = imageUrl.split('?');
  const marker = 'ik.imagekit.io/';
  const markerIndex = baseUrl.indexOf(marker);
  const pathStart = baseUrl.indexOf('/', markerIndex + marker.length);
  if (pathStart === -1) return imageUrl;
  return                `${baseUrl.slice(0,                pathStart)}/tr:e-
bgremove${baseUrl.slice(pathStart)}${queryString ? `?${queryString}` : ''}`;
};
```

Додаток Л. Лістинг розрахунку вартості оренди

```
export const RENTAL_START_HOUR = 8;
export const RENTAL_START_MINUTE = 0;
export const RENTAL_END_HOUR = 22;
export const RENTAL_END_MINUTE = 59;
export const OPTION_RATES = {
  deliveryPerKm: 30,
  returnPerKm: 30,
  childSeat: 150,
  bikeRack: 200,
  insuranceRate: 0.15,
};
export function applyRentalStartTime(date) {
  const d = new Date(date);
  d.setHours(RENTAL_START_HOUR, RENTAL_START_MINUTE, 0, 0);
  return d;
}
export function applyRentalEndTime(date) {
  const d = new Date(date);
  d.setHours(RENTAL_END_HOUR, RENTAL_END_MINUTE, 0, 0);
  return d;
}
export function countRentalDays(startDate, endDate) {
  const start = new Date(startDate.getFullYear(), startDate.getMonth(),
startDate.getDate());
  const end = new Date(endDate.getFullYear(), endDate.getMonth(),
endDate.getDate());
  return Math.round((end - start) / (1000 * 60 * 60 * 24)) + 1;
}
export function calculateRentalPrice(pricePerDay, startDate, endDate,
options = {}) {
  const days = countRentalDays(startDate, endDate);
  const basePrice = days * pricePerDay;
```

```
  const deliveryKm = options.delivery ? Math.max(0, Number(options.deliveryKm) || 0) : 0;
  const returnKm = options.returnElsewhere ? Math.max(0, Number(options.returnKm) || 0) : 0;
  const deliveryPrice = options.delivery ? deliveryKm * OPTION_RATES.deliveryPerKm : 0;
  const returnPrice = options.returnElsewhere ? returnKm * OPTION_RATES.returnPerKm : 0;
  const childSeatPrice = options.childSeat ? OPTION_RATES.childSeat : 0;
  const bikeRackPrice = options.bikeRack ? OPTION_RATES.bikeRack : 0;
  const insurancePrice = options.fullInsurance
    ? Math.round(basePrice * OPTION_RATES.insuranceRate)
    : 0;
  const optionsTotal =
    deliveryPrice + returnPrice + childSeatPrice + bikeRackPrice + insurancePrice;
  return {
    days,
    basePrice,
    deliveryPrice,
    returnPrice,
    childSeatPrice,
    bikeRackPrice,
    insurancePrice,
    optionsTotal,
    totalPrice: basePrice + optionsTotal,
  };
}
export function buildRentalOptionsPayload(pricing, formOptions) {
  return {
    delivery: {
      enabled: Boolean(formOptions.delivery),
      km: formOptions.delivery ? Math.max(0, Number(formOptions.deliveryKm) || 0) : 0,
```

```
      price: pricing.deliveryPrice,
    },
    return_elsewhere: {
      enabled: Boolean(formOptions.returnElsewhere),
      km:         formOptions.returnElsewhere         ?         Math.max(0,
Number(formOptions.returnKm) || 0) : 0,
      price: pricing.returnPrice,
    },
    child_seat: {
      enabled: Boolean(formOptions.childSeat),
      price: pricing.childSeatPrice,
    },
    bike_rack: {
      enabled: Boolean(formOptions.bikeRack),
      price: pricing.bikeRackPrice,
    },
    full_insurance: {
      enabled: Boolean(formOptions.fullInsurance),
      price: pricing.insurancePrice,
    },
  };
}
export function formatRentalTimeRange(startDate, endDate) {
  const start = applyRentalStartTime(startDate);
  const end = applyRentalEndTime(endDate);
  const fmt = new Intl.DateTimeFormat('uk-UA', {
    day: '2-digit',
    month: '2-digit',
    year: 'numeric',
    hour: '2-digit',
    minute: '2-digit',
  });
  return { start, end, label: `${fmt.format(start)} — ${fmt.format(end)}` };
}
```

Додаток М. Блок-схема алгоритму створення бронювання

Рисунок М – Блок-схема алгоритму створення бронювання

Додаток Н. Схема програмної взаємодії частин web-сайту

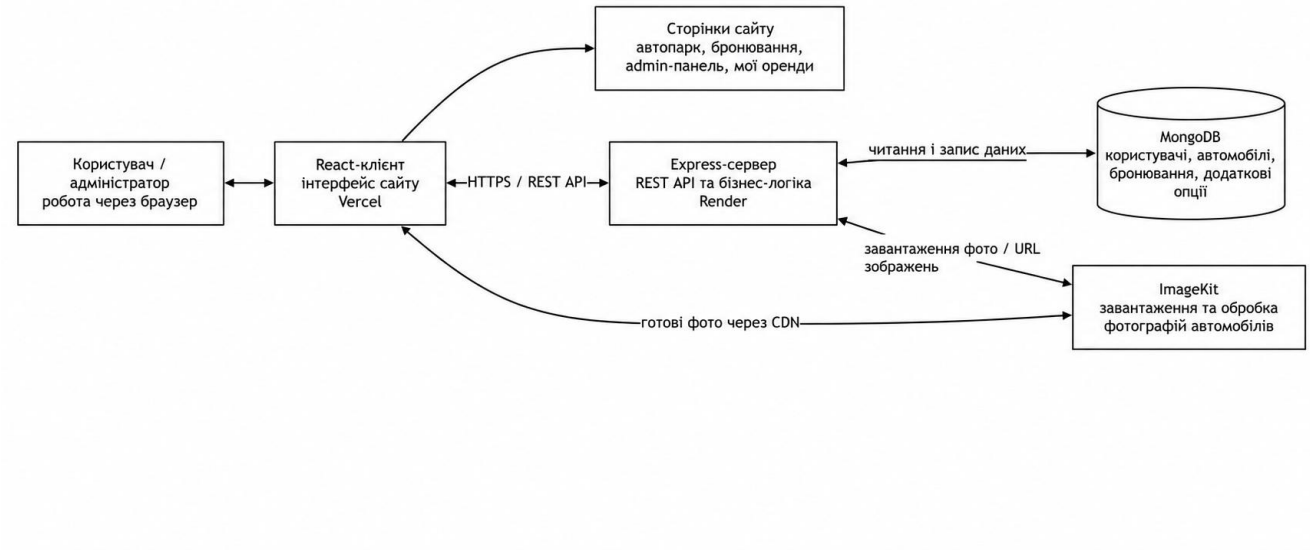
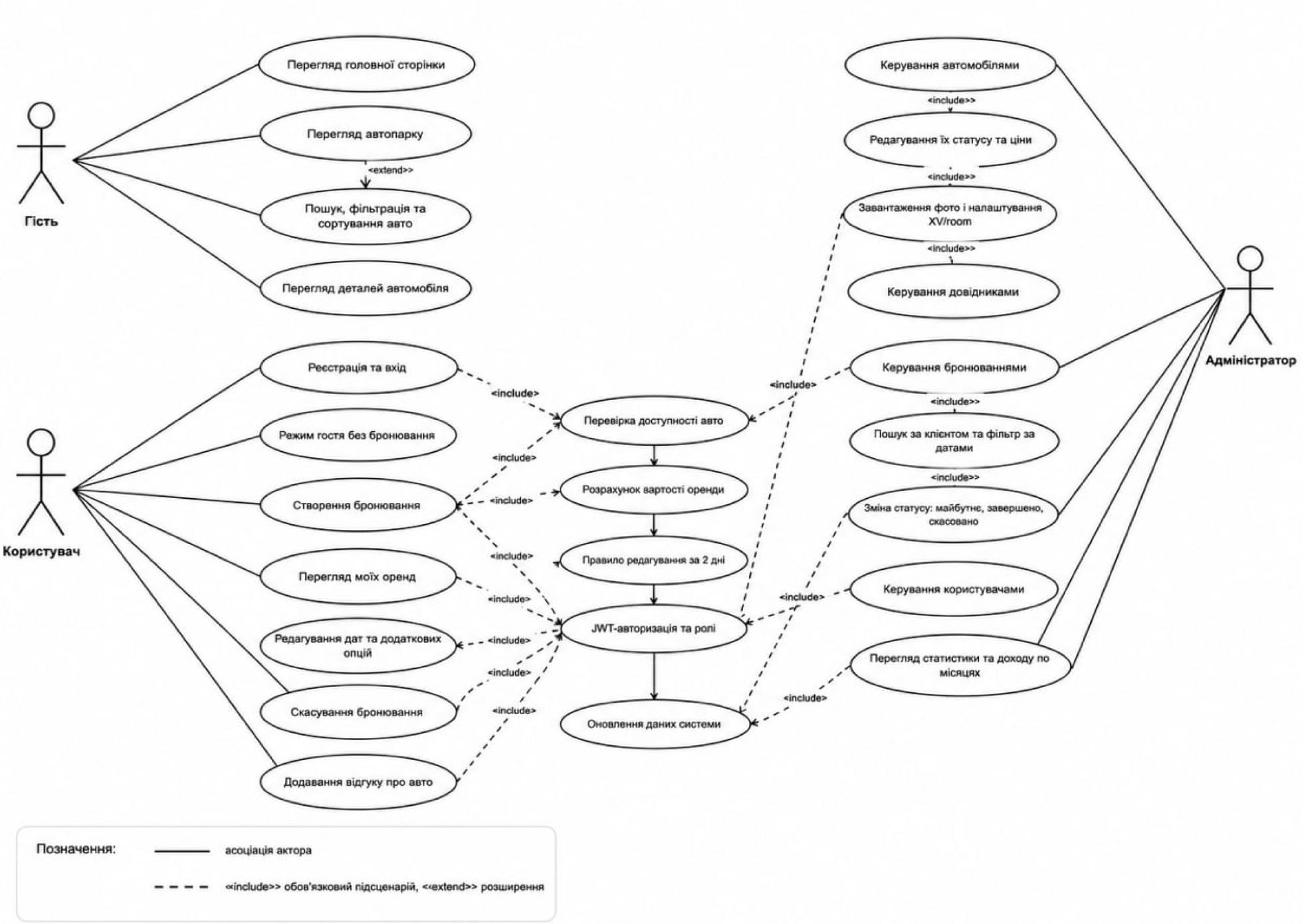


Рисунок Н – Програмна взаємодія частин web-сайту

UML-діаграма варіантів використання програми

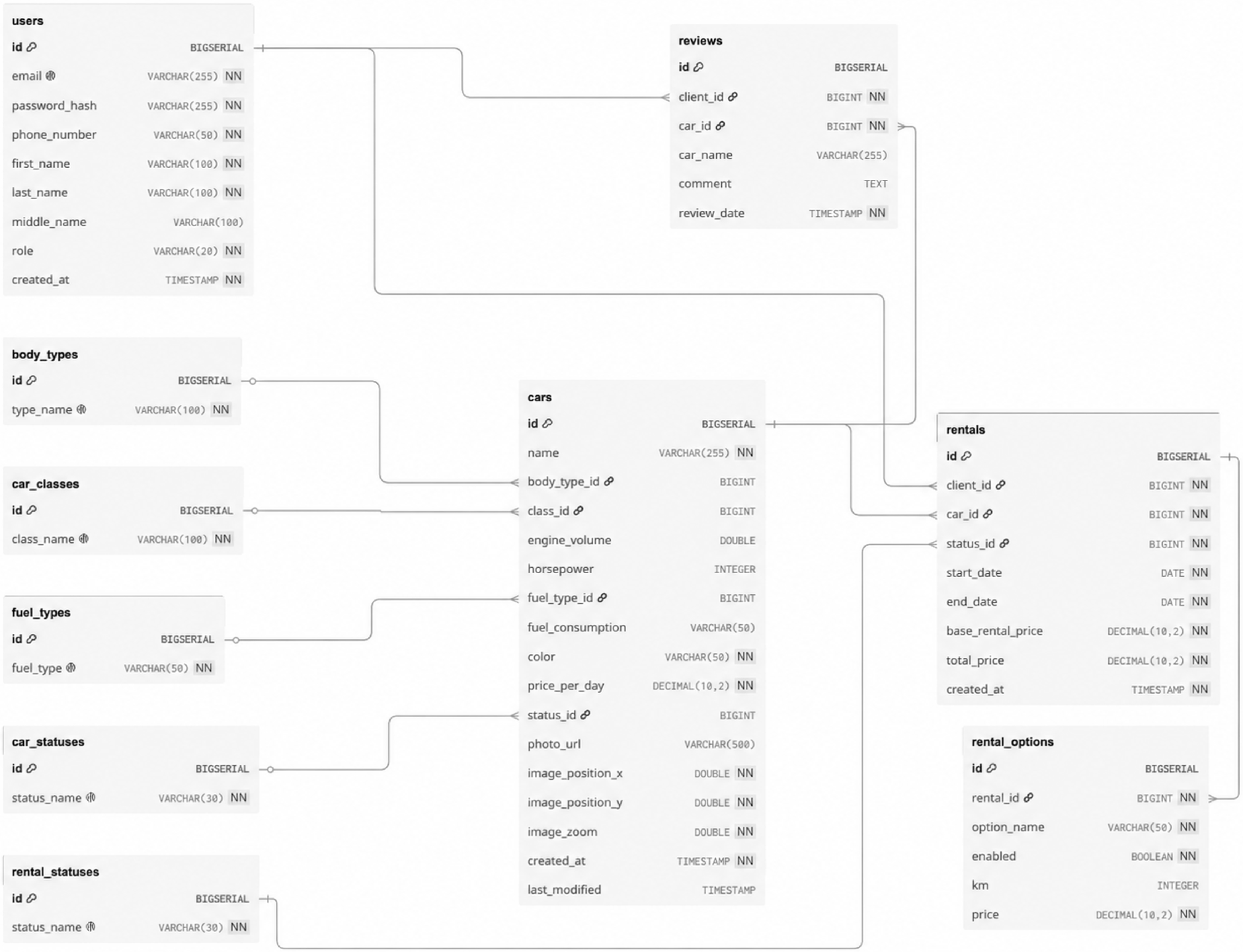


Зм.	Арк.	№ докум.	Підп.	Дата	2026.КВР.122.4.21.02.00.00 ДВ
Розроб.		Бридун М. В.			Розробка вебзастосунку для автоматизації управління автопарком «Burundiuk Garage»
Перев.		Райчик Г. І.			UML-діаграма варіантів використання програми
Т.контр.					Літ. / Маса / Масштаб
Рецензент					Аркуш 1 / Аркушів 1
Н.контр.		Приймак В.А.			ВСП ТФК ТНТУ КН-421 м. Тернопіль
Затв.					

Копіював Формат А1

Перш. застос. | Спрад. № | Підп. і дата | Інв. № дубл. | Взам. інв. № | Підп. і дата | Інв. № орігін.

ER-діаграма бази даних вебсайту

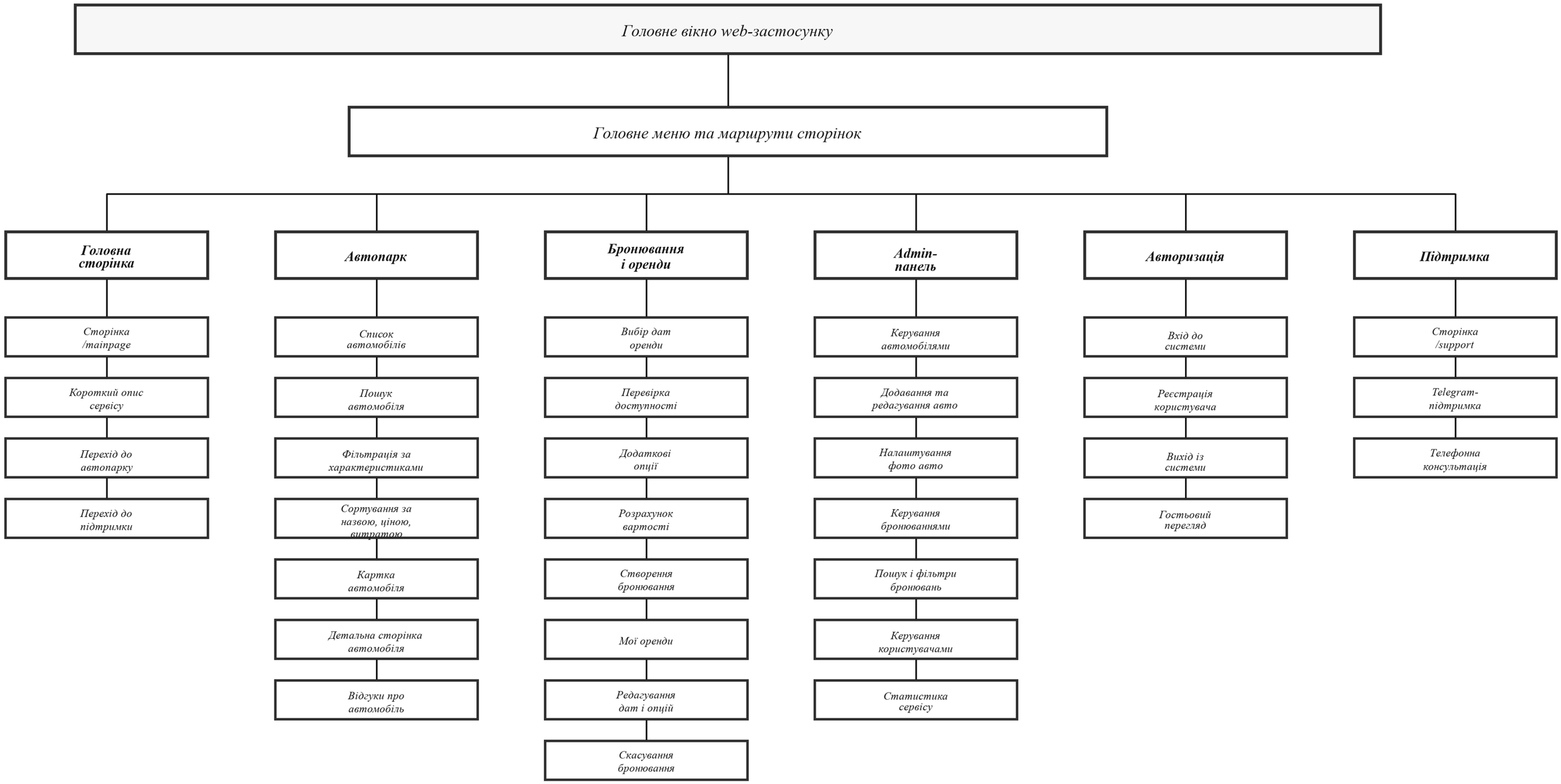


Зм.	Арк.	№ докум.	Підп.	Дата	2026.КВР.122.4.21.02.00.00 БД		
Розроб.		Бридун М. В.			Розробка вебзастосунку для автоматизації управління автопарком «Burundiuk Garage»	Літ.	Маса	Масштаб
Перевір.		Радчик Г. І.			ER-діаграма бази даних вебсайту			
Т.контр.						Аркуш	Аркушів 1	
Рецензент						ВСП ТФК ТНТУ КН-421		
Н.контр.		Приймак В.А.				м. Тернопіль		
Затв.								

Копіював Формат А1

Перш. застос. | Довід. № | Інв. № дубл. | Взам. інв. № | Підп. і дата | Інв. № орігін.

Структурна схема головної сторінки сайту



Доступ: гість переглядає сайт; користувач бронює і керує своїми орендами; адміністратор керує даними сервісу

Зм.	Арк.	№ докум.	Підп.	Дата	2026.КВР.122.421.02.00.00 СС		
Розроб.		Брайдун М. В.			Розробка вебзастосунку для автоматизації управління автопарком «Bigrindik Garage»	Літ.	Маса	Масштаб
Перев.		Радчик Г. І.						
Т.контр.					Схема структурна	Аркуш	Аркушів 1	
Рецензент						ВСП ТФК ТНТУ КН-421		
Н.контр.		Приймак В.А.				м. Тернопіль		
Затв.								

Копіював Формат А1

Перш. застос.

Справ. №

Підп. і дата

Інв. № дубл.

Взам. інв. №

Підп. і дата

Інв. № оригін.

Таблиця техніко-економічних показникі

№	Показник	Одиниці вимірювання	Значення
1	Платформа	–	Web
2	Інтерфейс	–	Вебзастосунок для оренди автомобілів
3	Мова програмування	–	JavaScript
4	Фреймворки та бібліотеки	–	React, Vite, Express, Tailwind CSS
5	База даних	–	MongoDB
6	Сервіс обробки зображень	–	ImageKit
7	Загальна трудомісткість розробки	год.	170
8	Собівартість	грн	87 388
9	Ціна	грн	136 325
10	Чиста теперішня вартість	грн	37 452
11	Термін окупності	рік	1,74