МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА СИСТЕМИ БРОНЮВАННЯ АВТОМОБІЛІВ ЗАСОБАМИ REACT NATIVE, NEXT.JS ТА FIREBASE»**

Виконав: студент 2 курсу, групи 8.1214-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
О.А. Базилевський
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент Директор ТОВ «Голден-Тім» Калін М.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Базилевському Олексію Андрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка системи бронювання автомобілів засобами React Native, Next.js та Firebase

керівник роботи Лимаренко Юлія Олексіївна, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 326-с

2. Строк подання студентом роботи 03.12.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд тенденцій платформ та технологій.
2. Проєктування системи.
3. Реалізація та тестування системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	07.03.2025	
2.	Збір вихідних даних.	23.04.2025	
3.	Обробка методичних та теоретичних	05.06.2025	
	джерел.		
4.	Розробка першого та другого розділу.	10.09.2025	
5.	Розробка третього розділу.	24.11.2025	
6.	Оформлення та нормоконтроль	01.12.2025	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	17.12.2025	

Студент _____ О.А. Базилевський
(підпис) (ініціали та прізвище)

Керівник роботи _____ Ю.О. Лимаренко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка системи бронювання автомобілів засобами React Native, Next.js та Firebase»: 42 с., 8 рис., 9 джерел, 1 додаток.

ВЕБЗАСТОСУНОК, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, МОБІЛЬНИЙ ЗАСТОСУНОК, ОРЕНДА АВТОМОБІЛІВ, СИСТЕМА БРОНЮВАННЯ, ХМАРНІ ТЕХНОЛОГІЇ, FIREBASE, NEXT.JS, REACT NATIVE.

Об’єкт дослідження – процес розробки мобільних та веборієнтованих інформаційних систем для онлайн-бронювання автомобілів.

Мета роботи – розробка та реалізація системи бронювання автомобілів із використанням сучасних веб- та мобільних технологій, що забезпечує зручний інтерфейс для користувачів та ефективну взаємодію з серверною частиною.

Методи дослідження – аналіз наукових і технічних джерел з теми роботи, методи проєктування інформаційних систем, об’єктно-орієнтований підхід, методи моделювання програмного забезпечення, практичні методи розробки та тестування програмних продуктів.

У кваліфікаційній роботі розглянуто процес проєктування та розробки системи бронювання автомобілів. Проведено аналіз предметної області та існуючих аналогів, визначено функціональні та нефункціональні вимоги до системи. Спроєктовано архітектуру програмного забезпечення та реалізовано клієнтську частину у вигляді мобільного застосунку з використанням React Native та вебзастосунку на основі Next.js. Для зберігання даних та організації серверної логіки використано хмарну платформу Firebase. Розроблена система забезпечує можливість перегляду доступних автомобілів, здійснення бронювання та взаємодії користувача з сервісом у зручному форматі.

SUMMARY

Master’s qualifying paper “Development of a Car Booking System with Mobile and Web Versions Using React Native, Next.js and Firebase”: 42 pages, 8 figures, 9 references, 1 supplement.

WEB APPLICATION, CLIENT-SERVER ARCHITECTURE, MOBILE APPLICATION, CAR RENTAL, BOOKING SYSTEM, CLOUD TECHNOLOGIES, FIREBASE, NEXT.JS, REACT NATIVE.

Object of the research is the process of developing mobile and web-oriented information systems for online car booking.

Purpose of the thesis is to develop and implement a car booking system using modern web and mobile technologies that provides a user-friendly interface and efficient interaction with the server-side components.

Research methods include the analysis of scientific and technical sources related to the research topic, information system design methods, the object-oriented approach, software modeling methods, as well as practical methods of software development and testing.

This qualification thesis examines the process of designing and developing a car booking system. An analysis of the subject area and existing analogues was carried out, and the functional and non-functional requirements of the system were defined. The software architecture was designed, and the client-side components were implemented in the form of a mobile application using React Native and a web application based on Next.js. The Firebase cloud platform was used for data storage and the implementation of server-side logic. The developed system enables users to view available cars, make bookings, and interact with the service in a convenient and efficient manner.

ЗМІСТ

Завдання на кваліфікаційну роботу ……..…..2
Реферат 4
Summary 5
Вступ..... 8
1 Огляд тенденцій, платформ та технологій 10
1.1 Сучасні тенденції у розвитку P2P-оренди та вебплатформ..... 10
1.2 Next.js як основа вебклієнта та його особливості..... 11
1.3 Firebase як serverless бекенд: Authentication, Firestore та Security 11
1.4 Порівняння підходів до кросплатформенності та вибір технологій . 12
1.5 Аналіз конкурентів і бенчмарків ринку..... 13
1.5.1 Getmancar (B2C)..... 13
1.5.2 NarsCars (локальний B2C)..... 13
1.5.3 Ukr-Prokat (агрегатор)..... 14
1.5.4 Turo (міжнародний P2P-бенчмарк) 14
1.5.5 Узагальнення 14
1.6 Технології для реалізації ключових модулів CarShare 14
1.6.1 Каталог, пошук і фільтрація 14
1.6.2 Аутентифікація та профіль користувача 15
1.6.3 Оголошення автомобілів (CRUD) та медіаконтент..... 15
1.6.4 Бронювання та перевірка доступності..... 15
1.6.5 «Обране» та оптимістичні оновлення..... 16
1.6.6 Безпека, продуктивність і масштабування..... 16
2 Проєктування системи..... 17
2.1 Архітектура системи..... 17
2.1.1 Загальний підхід до архітектури 17
2.1.2 Рівні архітектури (Layered Architecture)..... 17
2.1.3 Компонентна модель та маршрутизація..... 18

2.1.4 Патерни керування станом 19
2.1.5 Обробка помилок та UX-повідомлення 19
2.1.6 Підхід до кросплатформенності (Web + React Native) 19
2.2 Структура бази даних 20
2.2.1 Вибір СУБД та модель даних 20
2.2.2 Колекції та документи 20
2.2.3 Відношення між сутностями 22
2.2.4 Обмеження, пов’язані з base64 зображеннями 22
2.3 Діаграма варіантів використання 23
2.3.1 Актори системи 23
2.3.2 Перелік основних прецедентів 23
2.4 Діаграма послідовностей 24
2.4.1 Реєстрація користувача (email/password) 24
2.4.2 Додавання автомобіля (оголошення) 25
2.4.3 Бронювання автомобіля 26
2.4.4 Підтвердження бронювання власником 27
2.5 Діаграма класів (логічна модель) 28
2.6 Діаграма компонентів 30
2.7 Діаграма розгортання 31
3 Реалізація та тестування 33
3.1 Опис технологічного стеку 33
3.1.1 Frontend (Web) 33
3.1.2 Backend (Serverless) 33
3.2 Структура проєкту 33
3.3 Реалізація основних модулів 34
3.4 Тестування системи 35
3.5 Керівництво користувача (Web-версія) 37
Висновки 39
Перелік посилань 40
Додаток А Приклад структури документів Firestore 41

ВСТУП

Розвиток цифрових сервісів та зростання попиту на “sharing economy” формують нові вимоги до онлайн-платформ, які мають забезпечувати швидкий пошук, прозорі умови та зручне бронювання послуг у декілька кліків. Одним із практичних напрямів такого підходу є peer-to-peer оренда автомобілів, коли приватні власники напряму здають авто в оренду, а користувачі отримують ширший вибір та нижчу вартість у порівнянні з класичними прокатними сервісами. Саме розробці кросплатформенної системи бронювання автомобілів присвячена дана кваліфікаційна робота.

Актуальність теми полягає у необхідності створення зручного та надійного інструменту для P2P-оренди авто, який поєднує сучасний веб-інтерфейс, безпечну аутентифікацію, перевірені механізми керування даними та можливість подальшого масштабування на мобільні платформи. Використання Next.js дає змогу побудувати продуктивний веб-застосунок із підтримкою SSR та оптимізацією завантаження, а Firebase (Authentication, Firestore, Security Rules) — реалізувати serverless бекенд із керованою безпекою та масштабованістю. Додатковою перевагою є архітектура кросплатформенності (web та мобільні клієнти на базі React Native) за рахунок спільної моделі даних, типів та бізнес-логіки.

Мета роботи полягає в розробці кросплатформенної системи бронювання автомобілів CarShare, що забезпечує повний цикл взаємодії між орендарями та власниками: перегляд каталогу, фільтрацію, перегляд деталей, керування обраним, створення заявок на бронювання, а також управління оголошеннями та бронюваннями з різними статусами.

Завдання роботи:

– провести аналіз предметної області P2P-оренди автомобілів та сформувати вимоги до системи;
– обрати архітектурні підходи та технології реалізації (Next.js,

Firebase, React Query, Tailwind CSS, TypeScript) з урахуванням продуктивності та безпеки;

– спроєктувати модель даних та правила доступу (Firestore Security Rules) для ключових сутностей;
– реалізувати модулі аутентифікації та управління профілем;
– реалізувати модулі каталогу, пошуку та фільтрації, сторінки деталей оголошення та профілю власника;
– реалізувати механізм «обраного» з синхронізацією даних та оптимістичними оновленнями інтерфейсу;
– виконати тестування основних сценаріїв і оцінити продуктивність, UX та коректність обмежень безпеки.

Об’єкт дослідження: процес проєктування та розробки кросплатформенної системи бронювання автомобілів на основі Next.js та Firebase.

Предмет дослідження: методи, технології та архітектурні рішення, що застосовуються для створення P2P-сервісу бронювання (SSR, керування серверним станом, безпечна аутентифікація та авторизація в Firestore, оптимізація роботи з даними, кросплатформенність з React Native).

Методи дослідження: у роботі використано методи аналізу вимог та предметної області, проєктування структури даних і політик доступу, об’єктно-орієнтованого проєктування, розробки клієнтських інтерфейсів, тестування користувацьких сценаріїв, а також аналізу продуктивності та оптимізації взаємодії з Firestore із застосуванням кешування та інвалідації в React Query.

Структура роботи: кваліфікаційна робота складається зі вступу, розділу з оглядом тенденцій, платформ та технологій, розділу про проєктування системи (архітектура та модель даних), розділу реалізації та тестування, висновків, переліку посилань і додатків. Додатки містять додаткові матеріали, зокрема фрагменти коду, схеми та ілюстрації інтерфейсу.

1 ОГЛЯД ТЕНДЕНЦІЙ, ПЛАТФОРМ ТА ТЕХНОЛОГІЙ

1.1 Сучасні тенденції у розвитку P2P-оренди та вебплатформ

Розвиток цифрових сервісів та зростання попиту на сервіси спільного використання ресурсів (sharing economy) стимулюють появу платформ, які забезпечують швидкий пошук, прозорі умови та онлайн-бронювання без зайвих офлайн-процедур. У сфері оренди автомобілів особливо актуальною є модель peer-to-peer (P2P), де приватні власники напряму пропонують свої авто, а платформа виконує роль посередника, що забезпечує каталог, правила взаємодії, контроль доступів і зручні сценарії бронювання. Сучасні тенденції, що впливають на проєктування CarShare, можна узагальнити так:

- Mobile-first та кросплатформенність: користувачі очікують однаково зручної роботи як у браузері, так і в мобільному клієнті – це потребує уніфікації моделі даних, типів та бізнес-правил для web і mobile;
- продуктивність і SEO: публічні сторінки каталогу й деталей оголошень мають швидко завантажуватися та коректно індексуватися пошуковими системами, що робить релевантними SSR-підходи та оптимізацію медіа;
- Serverless-архітектура: для скорочення часу розробки та спрощення масштабування застосовуються керовані сервіси, які беруть на себе інфраструктурні задачі (аутентифікація, база даних, правила доступу);
- керування серверним станом: кешування, інвалідація та оптимістичні оновлення підвищують відчутну швидкість UI та зменшують кількість повторних запитів;
- безпека та довіра: ключові обмеження повинні перевірятися на бекенді, оскільки клієнтський код не є довіреним, тому авторизація та інваріанти моделі реалізуються через правила доступу та контроль структури даних.

Таким чином, для CarShare пріоритетними є швидкодія, безпека, масштабованість і готовність до розвитку мобільних клієнтів.

1.2 Next.js як основа вебклієнта та його особливості

Next.js [3] є React-фреймворком, який надає інструменти для створення продуктивних веб-застосунків з підтримкою SSR та оптимізацією на рівні маршрутизації і збірки. Для платформи бронювання це важливо через залежність від швидкого доступу до каталогу та сторінок оголошень.

Основні можливості Next.js, релевантні для CarShare:

- SSR для покращення швидкості першого відображення контенту та SEO;
- автоматичний code splitting по маршрутах для зменшення розміру стартового bundle;
- компонентний підхід React [4] для побудови повторно використовуваних UI-блоків (картки авто, форми, фільтри);
- TypeScript для типобезпеки та стабільності доменної моделі;
- Tailwind CSS [8] як utility-first підхід для швидкої розробки адаптивного інтерфейсу.

Для керування даними, отриманими з Firestore, застосовується React Query, що забезпечує кешування, повторні спроби, інвалідацію та оптимістичні оновлення.

1.3 Firebase як serverless бекенд: Authentication, Firestore та Security Rules

Firebase [1] дозволяє побудувати serverless бекенд і концентруватися на бізнес-логіці та UX. Для CarShare ключовими є Firebase Authentication [2], Firestore та Firestore Security Rules.

Firebase Authentication забезпечує реєстрацію та вхід користувачів (email/пароль, Google OAuth), керує токенами доступу та надає унікальний ідентифікатор користувача (UID). Використання UID як «опорного ключа» дає змогу реалізувати узгоджену модель доступів до даних та спростити інтеграцію між web і mobile-клієнтами.

Firestore є документно-орієнтованою NoSQL базою даних, що підходить для доменної моделі маркетплейсу: колекції користувачів, автомобілів, бронювань та зв’язків (наприклад, «обране»). Важливими є можливості фільтрації та індексації для роботи каталогу, а також транзакцій/батч-операцій для узгоджених змін.

Security Rules реалізують авторизацію на рівні бази даних, забезпечуючи такі обмеження:

– користувач може змінювати лише власний профіль та власні оголошення;
– доступ до бронювань мають лише учасники угоди (орендар або власник авто);
– публічні дані каталогу можуть бути доступні для читання, а створення/редагування – лише авторизованим.

Це гарантує коректність доступів незалежно від того, з якого клієнта надходить запит.

1.4 Порівняння підходів до кросплатформенності та вибір технологій

React Native [9] є доцільним вибором для CarShare, оскільки web-клієнт уже побудований на React/TypeScript. Переваги:

– спільність підходів та можливість повторного використання типів і частини логіки;
– інтеграція з тим самим Firebase-бекендом;

– швидший розвиток функціоналу за рахунок єдиної екосистеми.

Flutter забезпечує високу продуктивність та гнучкий UI, однак потребує іншої мови (Dart) і екосистеми, що ускладнює спільність коду з web-частиною.

Нативний підхід дає максимальний контроль і інтеграцію з платформою, але значно збільшує витрати на розробку та підтримку через окремі кодові бази для iOS та Android.

Для CarShare найбільш доцільним є кросплатформенний розвиток на базі React Native, оскільки це підтримує єдину модель даних, спільні підходи до стану та інтеграцію з Firebase.

1.5 Аналіз конкурентів і бенчмарків ринку

Для визначення конкурентних вимог до CarShare проаналізовано типові рішення на українському ринку та міжнародний бенчмарк P2P-моделі.

1.5.1 Getmancar (B2C)

Сервіс орієнтований на B2C-модель із власним автопарком. Переваги полягають у стандартизованому сервісі та підтримці, проте недоліками часто є вища ціна й географічні обмеження.

1.5.2 NarsCars (локальний B2C)

Локальний сервіс з обмеженим покриттям і менш гнучкими онлайн-сценаріями. Недоліками є обмежений вибір та залежність від офлайн-процесів.

1.5.3 Ukr-Prokat (агрегатор)

Агрегатор пропозицій надає можливість порівняння, але не реалізує повного циклу онлайн-бронювання та взаємодії між сторонами.

1.5.4 Turo (міжнародний P2P-бенчмарк)

Turo демонструє зрілу P2P-модель з мобільними клієнтами та розвиненими механізмами довіри, але сервіс не працює в Україні та функціонує в інших юридичних і бізнес-умовах.

1.5.5 Узагальнення

CarShare займає нішу P2P-платформи, поєднуючи сучасний web-клієнт, serverless-бекенд та архітектурну готовність до розвитку мобільних застосунків.

1.6 Технології для реалізації ключових модулів CarShare

Технологічні рішення CarShare спрямовані на забезпечення повного циклу сценаріїв користувачів: каталог, оголошення, бронювання, профілі та «обране».

1.6.1 Каталог, пошук і фільтрація

Фільтрація реалізується через параметри (ціна, локація, категорія, тип палива, трансмісія, кількість місць) із можливістю збереження стану у URL.

React Query забезпечує кешування та узгоджене оновлення даних при зміні фільтрів.

1.6.2 Аутентифікація та профіль користувача

Аутентифікація будується на Firebase Authentication, а профіль користувача зберігається у Firestore та прив’язується до UID. Такий підхід спрощує контроль доступу в Security Rules та забезпечує узгодженість між web і mobile.

1.6.3 Оголошення автомобілів (CRUD) та медіаконтент

Створення й редагування оголошень реалізується через форми з валідацією [6] (React Hook Form + Zod). Для спрощення архітектури медіаконтент може зберігатися у форматі base64 у Firestore за умови контролю розміру та кількості зображень на клієнті.

1.6.4 Бронювання та перевірка доступності

Модуль бронювання включає вибір дат, розрахунок вартості, перевірку доступності авто та керування статусами. Узгодженість змін забезпечується транзакціями або батч-операціями Firestore, а доступ обмежується правилами безпеки.

1.6.5 «Обране» та оптимістичні оновлення

Механізм «обраного» реалізується як зв’язок між користувачем і оголошеннями та підтримує оптимістичні оновлення інтерфейсу для миттєвого UX з подальшою синхронізацією з Firestore.

1.6.6 Безпека, продуктивність і масштабування

Безпека забезпечується поєднанням Security Rules і валідації структури даних. Продуктивність досягається завдяки SSR у Next.js, кешуванню React Query та оптимізації Firestore-запитів (індекси, ліміти, мінімізація читань). Serverless-підхід Firebase дозволяє масштабувати систему без підтримки власної серверної інфраструктури.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Архітектура системи

2.1.1 Загальний підхід до архітектури

Система CarShare розробляється як кросплатформенний продукт, що складається з:

– web-клієнта (реалізовано) на базі Next.js (React) з адаптивною версткою та клієнтським керуванням станом через React Query;
– мобільного клієнта (React Native для Android/iOS) – у розробці, ще не релізнутий;
– спільного бекенда у вигляді керованих сервісів Firebase: Authentication + Firestore, які виконують роль централізованого джерела істини (Single Source of Truth) для даних, та забезпечують контроль доступу (Security Rules).

Таким чином, логічно система реалізує клієнт-серверну модель, але технічно бекенд організований за serverless-парадигмою: відсутній власний HTTP-сервер з бізнес-логікою, а операції виконуються через SDK до керованих сервісів.

2.1.2 Рівні архітектури (Layered Architecture)

Архітектуру web-клієнта можна описати як багатошарову:

– UI layer – React/Next.js сторінки та компоненти (src/app, src/components/ui, src/components/sections, src/components/layout) ;
– State/Data layer – React Query hooks + мутації (src/api/resources/*), що надають декларативний доступ до даних;

– Domain/Types layer – типи сутностей (авто, користувач, бронювання) (src/types/*);
– Infrastructure layer – Firebase SDK ініціалізація та утиліти (src/tools/firebase.ts, src/tools/auth.ts, src/tools/error.ts).

Подібне розбиття дозволяє локалізувати зміни (наприклад, змінювати правила формування запитів у actions.ts без правок UI), перевикористовувати доменні типи та частину data-логіки в мобільному застосунку та підтримувати розширення системи (відгуки, платежі, push-нотифікації) без масового переписування.

2.1.3 Компонентна модель та маршрутизація

У web-версії використано Next.js App Router, що дозволяє організувати сторінки як дерево директорій.

Основні сторінки:

– /каталог + фільтри пошуку;
– /cars/new – створення оголошення;
– /cars/[carId] – деталі авто + бронювання;
– /cars/[carId]/edit – редагування авто;
– /my-cars – оголошення користувача;
– /favorites – обране;
– /bookings – бронювання (мої та запити до моїх авто);
– /user/[userId] – профіль користувача;
– /login, /signup – автентифікація.

UI побудовано з використанням повторно-вживаних компонентів: картки авто, поля форми, кнопки, типографіка, секції сторінок.

2.1.4 Патерни керування станом

Стан у системі умовно поділяється на **серверний стан** (дані з Firestore: авто, бронювання, профілі), керується React Query, що забезпечує кешування, повторні спроби, інвалідацію, та узгоджене оновлення UI; і **локальний** UI-стан (відкритість модальних вікон, вибір фото в галереї, активна вкладка) – керується локальним React state.

Ключовий принцип: не дублювати серверний стан у локальний без необхідності. Дані, що приходять з Firestore, залишаються під контролем React Query, а компоненти підписуються на них через hooks.

2.1.5 Обробка помилок та UX-повідомлення

Для користувача критично важливо розуміти результат операцій (створення оголошення, бронювання, зміна статусу). Тому частина помилок обробляється через алерти/повідомлення (наприклад, коли не обрані дати), а помилки автентифікації мапляться на зрозумілі повідомлення (src/tools/error.ts); у довгих операціях показуються Loading/disabled-стани кнопок.

2.1.6 Підхід до кросплатформенності (Web + React Native)

Мобільна версія CarShare розглядається як окремий клієнт, що працює з тим самим Firebase-бекендом. На рівні проєктування передбачено:

– спільну модель даних (типи Car, User, Booking);
– спільні правила доступу (Firestore Security Rules) незалежно від платформи;
– однакові бізнес-правила (наприклад: лише власник може редагувати

авто; бронювання доступне лише авторизованим; бронювання читають лише сторони угоди).

2.2 Структура бази даних

2.2.1 Вибір СУБД та модель даних

Як сховище даних обрано Firebase Firestore – документо-орієнтовану NoSQL базу даних. Основними причинами вибору стала природна модель документів для оголошень та бронювань, просте масштабування на рівні сервісу (без адміністрування серверів), інтеграція з Firebase Authentication через uid та можливість налаштування доступу через Security Rules. При цьому, важливо зазначити про зручність інтеграції і для web, і для мобільного клієнта.

2.2.2 Колекції та документи

У базі даних використовуються основні колекції:

– users – дані користувачів;
– cars – оголошення автомобілів;
– bookings – заявки/бронювання.

Документ користувача містить:

– id: string (uid з Firebase Auth, використовується як id документа);
– name: string;
– email: string;
– avatarUrl: string | null (base64 або порожньо);
– phone: string | null;
– likedCars: string[] (список carId в обраному);

– myCars: string[] (опціонально; в поточній реалізації список авто користувача також може визначатися запитом по ownerId);
– reservations: string[] (список bookingId, якщо використовується);
– createdAt, updatedAt: Timestamp.

Документ оголошення:

– id: string (id документа);
– ownerId: string (посилання на users/{uid});
– title: string (згенерована назва, наприклад “Audi A6”);
– brand, model: string;
– year: number;
– pricePerDay: number (грн/день);
– location: string (наприклад «Київ, Хрещатик»);
– description: string;
– images: string[] (до 5 зображень у форматі base64);
– fuelType: "petrol" | "diesel" | "electric" | "hybrid";
– transmission: "manual" | "automatic";
– seats: number;
– category: "economy" | "comfort" | "business" | "premium" | "suv" | "minivan";
– features: string[] (опціонально);
– isAvailable: boolean;
– createdAt, updatedAt: Timestamp;
– rating, reviewsCount: number (опціонально; підготовлено під майбутній функціонал).

Документ бронювання:

– id: string (id документа);
– carId: string;
– renterId: string;
– ownerId: string;
– startDate, endDate: Timestamp;

– totalPrice: number;
– status: "pending" | "confirmed" | "active" | "completed" | "cancelled";
– message: string (опціонально);
– createdAt: Timestamp.

2.2.3 Відношення між сутностями

Firestorm є NoSQL, тому зв’язки реалізуються через ідентифікатори:

– cars.ownerId → users/{ownerId};
– bookings.carId → cars/{carId};
– bookings.renterId → users/{renterId};
– bookings.ownerId → users/{ownerId};
– users.likedCars[] містить id з cars.

2.2.4 Обмеження, пов’язані з base64 зображеннями

Вимога проєкту – не використовувати Firebase Storage, а зберігати фото як base64 у Firestore. Це створює ряд технічних обмежень, таких як: зростає розмір документа cars/{carId}; потрібна строгіша валідація розміру фото на клієнті; необхідно обмежувати кількість фото.

У поточній реалізації застосовано політики до 5 фото, до 500KB на файл (до конвертації), з повідомленням користувачу при перевищенні та перше фото вважається головним.

Це дозволяє зберігати оголошення атомарно: дані авто + фото в одному документі.

2.3 Діаграма варіантів використання

2.3.1 Актори системи

У системі виділяються такі актори анонімний користувач – відвідувач без авторизації. Авторизований користувач (орендар) – може бронювати та вести обране. Авторизований користувач (власник) – може публікувати оголошення та обробляти запити на бронювання.

Примітка: одна фізична особа може одночасно виступати і як орендар, і як власник, залежно від сценарію.

2.3.2 Перелік основних прецедентів

Для анонімного користувача:

– перегляд каталогу;
– фільтрація та пошук;
– перегляд сторінки авто;
– перегляд профілю власника;
– перехід до авторизації/реєстрації.

Для авторизованого користувача (орендар):

– вхід/вихід;
– керування профілем;
– додавання/видалення авто з обраного;
– створення бронювання;
– перегляд «Мої бронювання».

Для авторизованого користувача (власник):

– створення оголошення;
– редагування/видалення оголошення;
– зміна доступності авто;

– перегляд «Запити на оренду»;
– підтвердження/відхилення/завершення бронювання.

2.4 Діаграма послідовностей

Для ключових сценаріїв наведено текстові діаграми послідовностей у форматі Mermaid (для Markdown). При конвертації у Word вони зберігаються як структурований текст.

2.4.1 Реєстрація користувача (email/password)

Діаграма послідовності, яка представлена на рисунку 2.1, ілюструє процес реєстрації нового користувача в системі за допомогою електронної пошти та пароля.

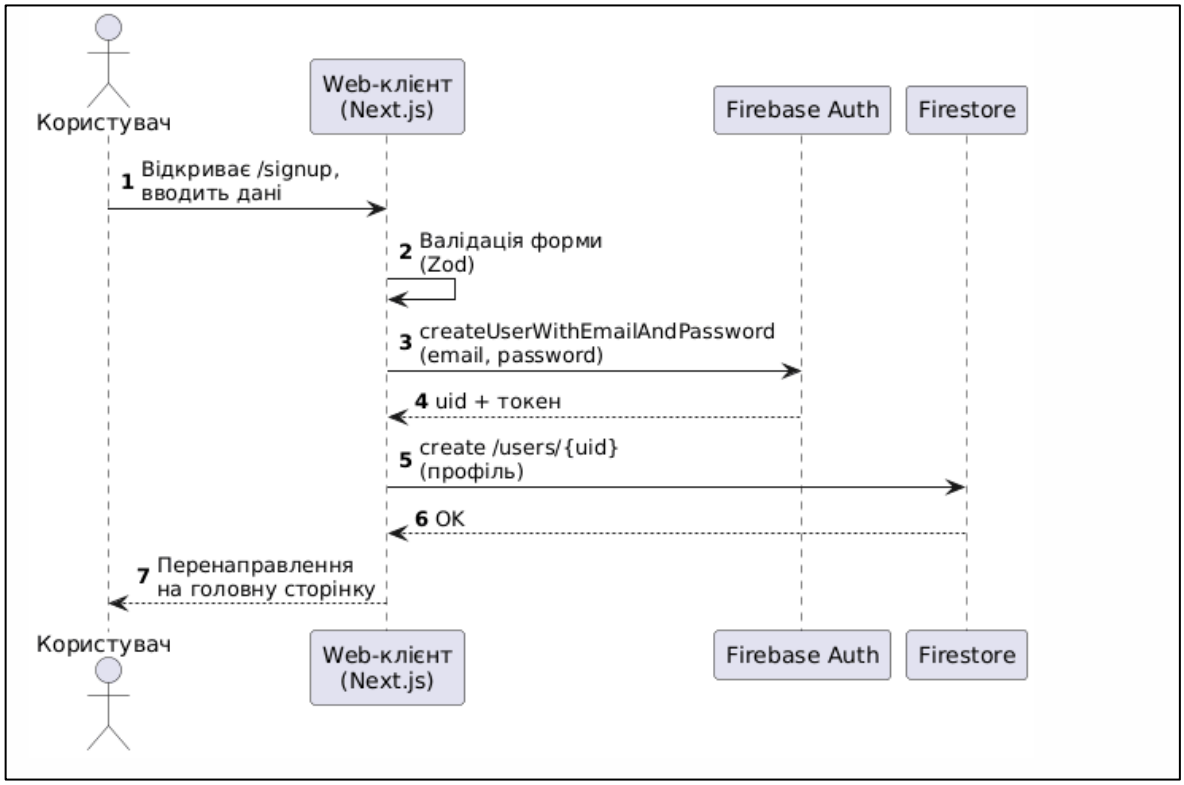


Рисунок 2.1 – Діаграма реєстрації нового користувача

Процес починається з того, що користувач відкриває сторінку реєстрації /signup та вводить свої облікові дані. На стороні web-клієнта, реалізованого з використанням фреймворку Next.js, виконується первинна валідація введених даних за допомогою бібліотеки Zod [7], що дозволяє запобігти надсиланню некоректної інформації на сервер.

Після успішної валідації web-клієнт викликає метод сервісу Firebase Authentication для створення нового облікового запису. У відповідь система автентифікації повертає унікальний ідентифікатор користувача (uid) та токен доступу.

Далі клієнтська частина ініціює створення профілю користувача в базі даних Cloud Firestore, де зберігається розширена інформація про користувача. Після успішного завершення операції користувач автоматично перенаправляється на головну сторінку системи.

Даний сценарій забезпечує безпечну та послідовну реєстрацію користувачів із чітким розмежуванням відповідальностей між клієнтською та серверною частинами.

2.4.2 Додавання автомобіля (оголошення)

На діаграмі послідовностей (див. рис. 2.2) представлено сценарій створення нового оголошення про автомобіль власником.

Власник автомобіля переходить на сторінку створення оголошення /cars/new, де завантажує фотографії транспортного засобу та заповнює форму з основними характеристиками. На стороні web-клієнта здійснюється перевірка кількості та розміру зображень, а також їх підготовка до збереження (конвертація у відповідний формат).

Після заповнення форми відбувається валідація введених даних, що гарантує цілісність інформації. Далі клієнт ініціює створення нового документа в колекції cars бази даних Firestore, де зберігається інформація про автомобіль разом з ідентифікатором власника.

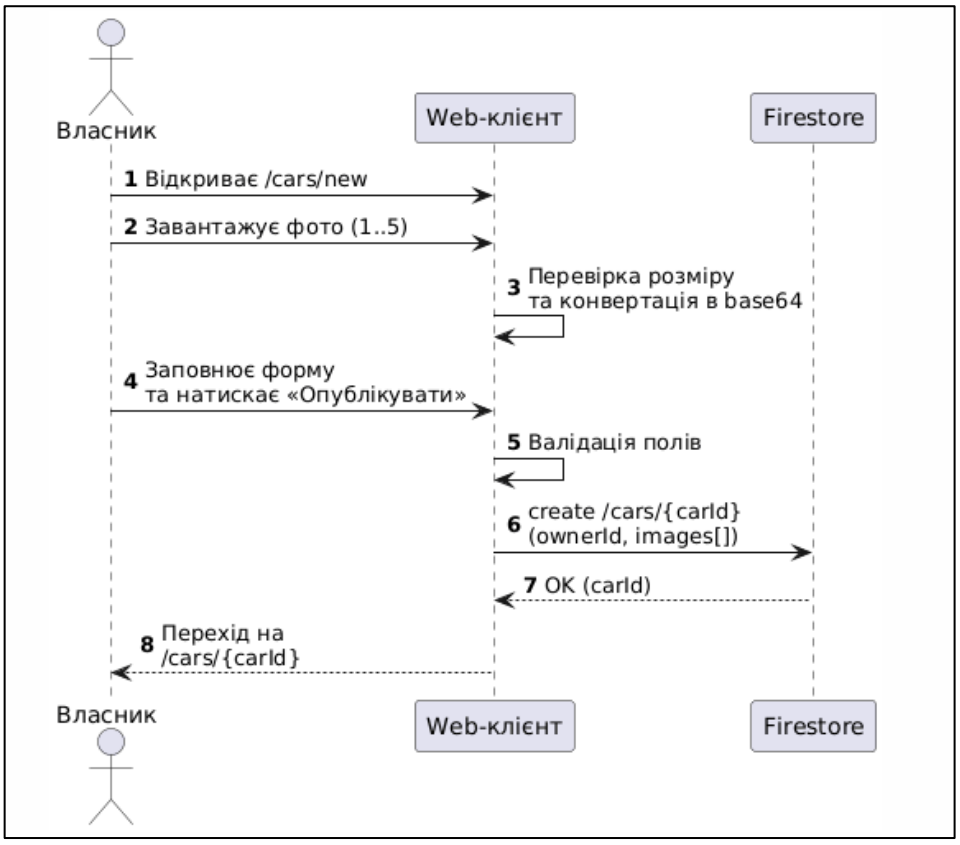


Рисунок 2.2 – Діаграма створення нового оголошення

У разі успішного створення оголошення користувач перенаправляється на сторінку перегляду доданого автомобіля.

2.4.3 Бронювання автомобіля

Сценарій бронювання автомобіля орендарем представлено на рисунку 2.3.

Орендар відкриває сторінку конкретного автомобіля, обирає дати початку та завершення оренди, після чого система автоматично розраховує загальну вартість бронювання на основі ціни за добу.

Після підтвердження дії клієнт надсилає запит до бази даних Firestore для створення нового запису бронювання зі статусом pending.

Такий підхід дозволяє чітко відокремити етап створення бронювання від його підтвердження власником, що підвищує гнучкість бізнес-логіки системи.

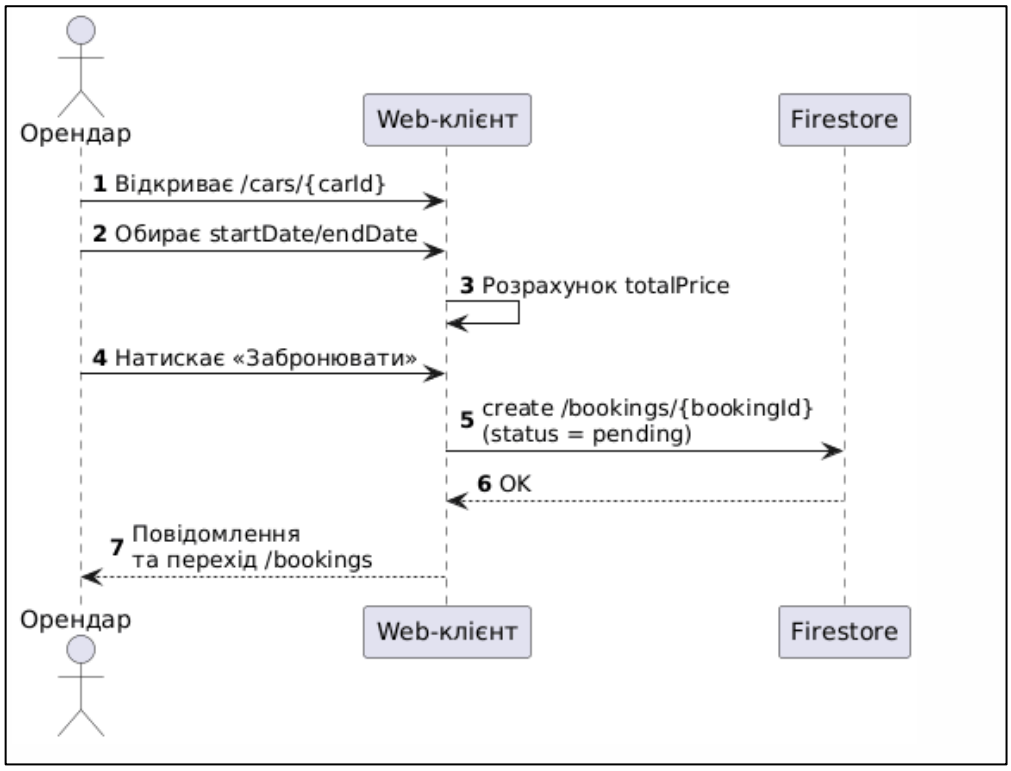


Рисунок 2.3 – Діаграма бронювання автомобіля

Після збереження бронювання користувач отримує відповідне повідомлення та перенаправляється до розділу зі своїми бронюваннями.

2.4.4 Підтвердження бронювання власником

Діаграма послідовності, яка представлена на рисунку 2.4, ілюструє процес підтвердження бронювання власником автомобіля.

Власник відкриває розділ перегляду запитів на оренду, де відображаються всі бронювання зі статусом очікування. Після натискання кнопки підтвердження web-клієнт надсилає запит до Firestore на оновлення статусу відповідного бронювання на confirmed.

Після успішного оновлення даних інтерфейс користувача автоматично оновлюється, що дозволяє власнику одразу бачити актуальний стан бронювання. Даний механізм забезпечує прозору та зручну взаємодію між учасниками процесу оренди.

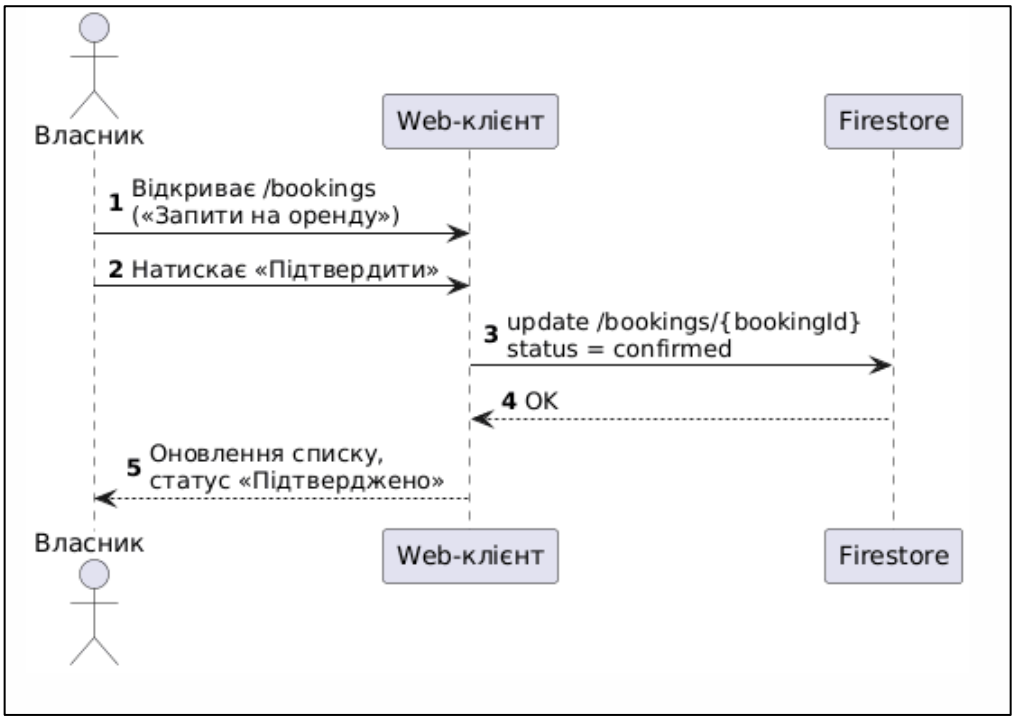


Рисунок 2.4 – Діаграма підтвердження бронювання власником

2.5 Діаграма класів (логічна модель)

У web-версії доменні «класи» реалізовано не як ООП-класи, а як TypeScript-типи та сервісні функції/хуки доступу до даних. Базові сутності (див. рис. 2.5): User, Car, Booking.

User описує профіль користувача (ідентифікатор, ім’я, контакти) та його списки: обрані авто (likedCars) і перелік власних оголошень (myCars).

Car описує оголошення автомобіля: належність власнику (ownerId), основні характеристики (марка/модель/рік/ціна тощо), параметри (тип пального/трансмісія/категорія), медіа/опис і ознаку доступності.

Booking фіксує бронювання: пов’язує автомобіль (carId) та учасників (власник ownerId і орендар renterId), містить період оренди (startDate, endDate), підсумкову вартість, статус і (за потреби) повідомлення.

Зв’язки:

– User 1 → N Car: один користувач може мати багато оголошень (зв’язок через ownerId);

– Car 1 → N Booking: один автомобіль може мати багато бронювань у часі (зв’язок через carId);
– User 1 → N Booking у двох ролях: користувач може бути власником або орендарем (окремі поля ownerId та renterId).

Розділення ролей спрощує вибірки «мої бронювання як власника/як орендаря» та відповідає логіці керування статусами і кабінету користувача.

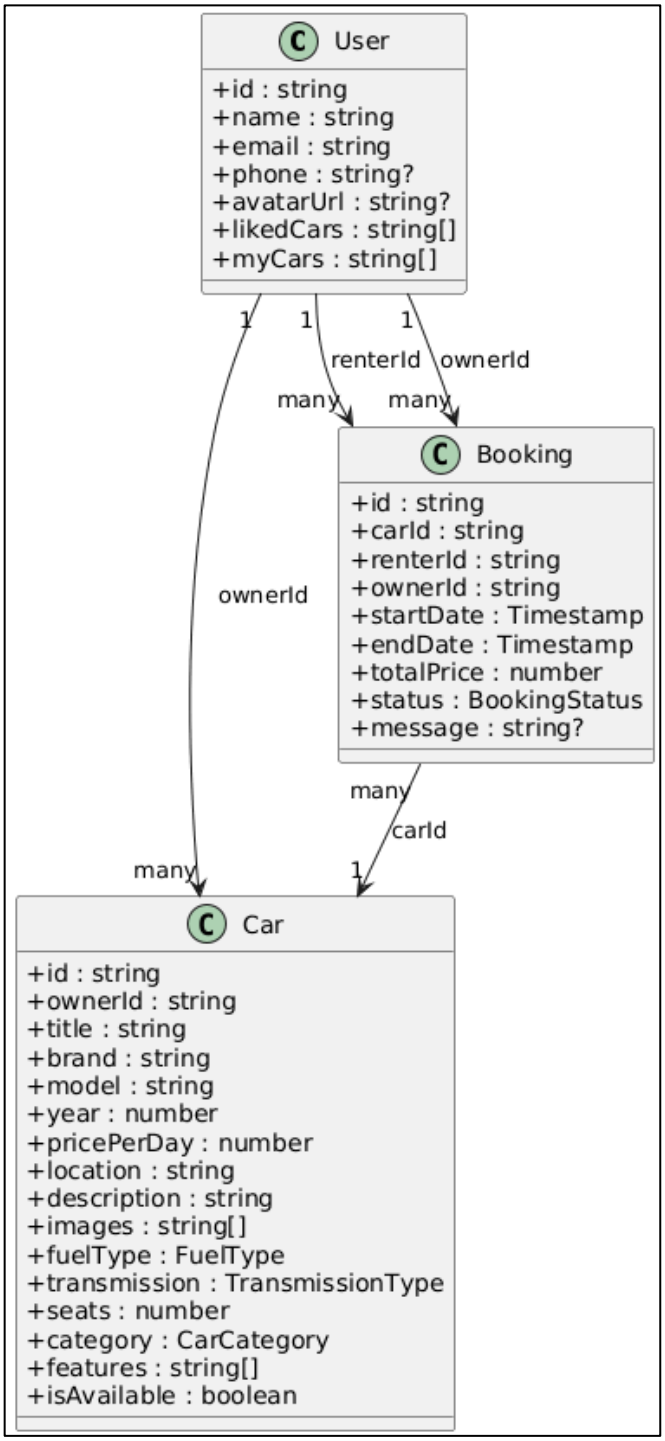


Рисунок 2.5 – Діаграма сутностей

2.6 Діаграма компонентів

Архітектура web-додатка організована шарами, щоб UI не залежав від конкретної реалізації сховища, а доступ до даних був централізований і повторно використовувався. Це відображено на рисунку 2.6.

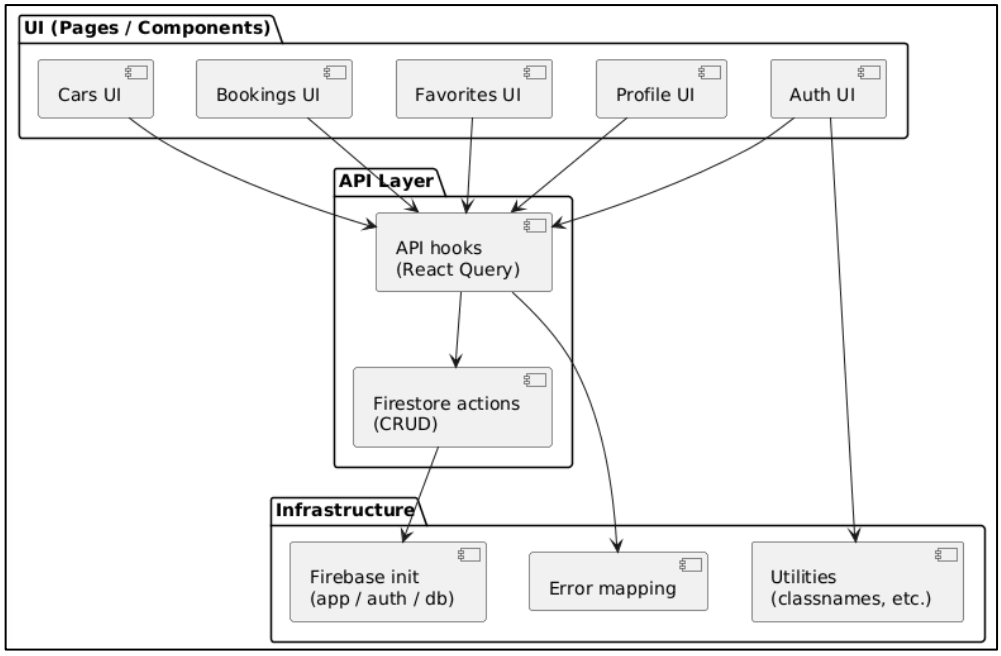


Рисунок 2.6 – Діаграма компонентів

UI layer (Pages/Components): сторінки та компоненти Next.js (каталог авто, картки, форми, профіль, обране, бронювання). UI відповідає за стан інтерфейсу, валідацію вводу на рівні форм і відображення результатів.

API layer (hooks, React Query): набір хуків (query/mutation), що інкапсулюють отримання/зміну даних, кешування, інвалідацію та обробку станів loading/error/success. UI взаємодіє з даними лише через цей шар.

Firestore actions (CRUD): низькорівневі функції для операцій читання/запису в Cloud Firestore (створення/оновлення оголошень, створення бронювань, зміна статусів, оновлення профілю, робота з обраним).

Infrastructure:

– Firebase init (app/auth/db): єдина ініціалізація Firebase SDK та доступ до Auth/Firestore;

– Error mapping: приведення помилок Firebase/Firestore до формату, зручного для UI (читабельні повідомлення, типізація);
– Utilities: спільні утиліти (classnames, форматування, допоміжні хелпери), не прив’язані до домену.

Підсумок: зміни в схемі даних або способі доступу локалізуються в actions/hooks, а UI залишається «тонким» і стабільним.

2.7 Діаграма розгортання

Розгортання web-версії включає три основні вузли (див. рис. 2.7):

– браузер користувача: виконує клієнтський код інтерфейсу (Web UI), ініціює операції через Firebase SDK;
– Vercel: хостинг Next.js-застосунку (доставка сторінок/статики/асетів). Користувач отримує застосунок по HTTPS;
– Firebase Auth – реєстрація/вхід, керування сесією;
– Cloud Firestore – зберігання доменних даних (профілі, авто, бронювання, обране).

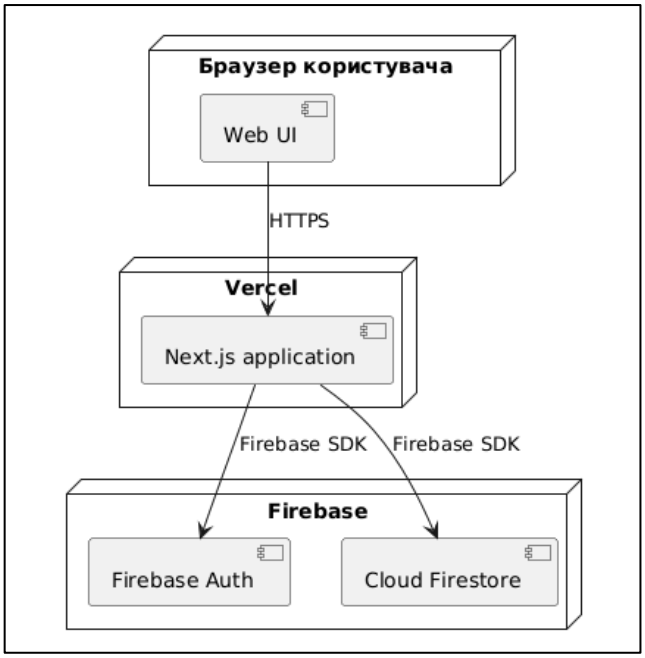


Рисунок 2.7

Взаємодія web-клієнта з Firebase відбувається через Firebase SDK по захищеному каналу, без окремого власного backend-сервера для CRUD-операцій. Аналогічна схема застосовується і для мобільного клієнта (React Native): застосунок на пристрої напряму використовує Firebase SDK для Auth/Firestore, забезпечуючи єдине джерело даних і спільні правила доступу на стороні Firestore (див. рис. 2.8).

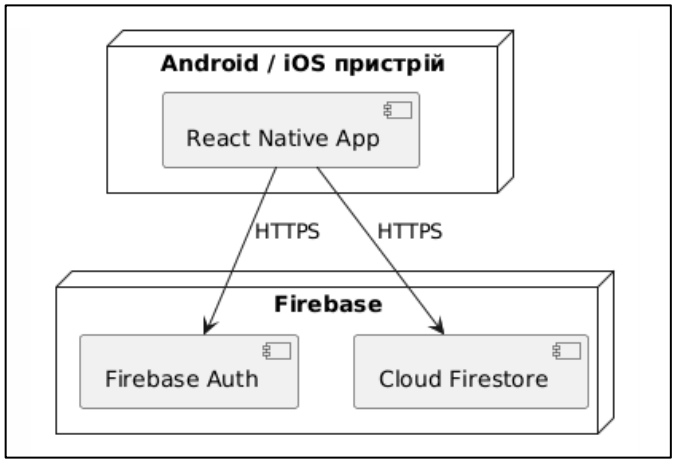


Рисунок 2.8

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис технологічного стеку

3.1.1 Frontend (Web)

Для розробки фронтенд частини використовувались:

– Next.js – маршрутизація, SSR, структура сторінок;
– React – компонентна модель UI;
– TypeScript – типобезпека та контроль контрактів даних;
– Tailwind CSS – швидка стилізація та консистентний дизайн;
– React Hook Form + Zod – форми та валідація;
– TanStack React Query [5] – кешування, запити та мутації.

3.1.2 Backend (Serverless)

Запити на авторизацію та отримання та редагування даних обробляються завдяки Firebase, зокрема:

– Firebase Authentication – email/password та Google OAuth;
– Cloud Firestore – документо-орієнтоване сховище;
– Firestore Security Rules – контроль доступу до даних.

3.2 Структура проєкту

У web-версії структура коду побудована навколо доменів та повторного використання компонентів.

Ключові каталоги:

– src/app/ – сторінки (маршрути);

– src/components/ – UI та секції;
– src/api/ – data-layer: hooks, actions, queryClient;
– src/types/ – типи домену;
– src/tools/ – інфраструктура та утиліти.

3.3 Реалізація основних модулів

Модуль автентифікації:
– реєстрація користувача;
– вхід (email/password, Google);
– вихід;
– створення профілю користувача в Firestore після реєстрації/першого входу.

Критичні моменти реалізації:
– uid з Firebase Auth використовується як id профілю в users;
– помилки автентифікації транслюються у зрозумілі повідомлення;
– після логіну користувач отримує доступ до захищених сторінок (додавання авто, бронювання, обране).

Модуль «Автомобілі»:
– створення оголошення з фото у base64;
– редагування/видалення оголошення власником;
– зміна доступності авто;
– відображення каталогу з фільтрами.

Реалізаційні акценти:
– в UI додано захист від undefined полів (наприклад, якщо images порожній);
– у Firestore запитах уникнено складних orderBy разом з where, щоб не примушувати проєкт залежати від композитних індексів;
– сортування (за потреби) виконується на клієнті.

Модуль «Бронювання»:

– створення бронювання з розрахунком суми;
– перегляд «Мої бронювання» (орендар);
– перегляд «Запити на оренду» (власник);
– зміна статусів (pending → confirmed → active → completed/cancelled) (статуси реалізують життєвий цикл заявки).

Модуль «Обране»:

– додати/видалити авто в обране;
– відобразити горизонтальний список швидкого перегляду;
– відобразити повний список обраних авто (обране зберігається як масив likedCars у документі користувача).

Модуль «Профіль користувача»:

– перегляд профілю;
– редагування власного профілю (ім’я, аватар, телефон);
– перегляд списку авто користувача;
– навігація до авто та до списків (обране, бронювання).

3.4 Тестування системи

Тестування в контексті цього проєкту включає ручне функціональне тестування (E2E сценарії) для ключових потоків; перевірку правил доступу (Security Rules) через реальні сценарії з різними ролями; регресійні перевірки UI після змін в компонентах (наприклад, після українізації текстів та зміни валюти на гривню).

TC-AUTH-01. Реєстрація email/password:

– відкрити /signup;
– ввести валідний email та пароль (>= 6 символів);
– натиснути «Зареєструватися».

Очікувано: користувач створений, профіль є у Firestore users/{uid}, виконано редирект на головну.

TC-CARS-01. Створення оголошення:

– авторизуватися;

– відкрити /cars/new;

– додати 1–5 фото (<= 500KB);

– заповнити обов’язкові поля;

– натиснути «Опублікувати».

Очікувано: створено cars/{carId}, сторінка деталей авто відображається без помилок.

TC-CARS-02. Заборона редагування чужого авто:

– авторизуватися як користувач A;

– відкрити авто користувача B;

– спробувати виконати редагування/видалення.

Очікувано: UI не показує дії редагування; Firestore Rules забороняють update/delete.

TC-BOOK-01. Створення бронювання:

– авторизуватися як орендар;

– відкрити /cars/{carId} чужого авто;

– вибрати дати;

– натиснути «Забронювати».

Очікувано: створено bookings/{bookingId} зі статусом pending, відображено повідомлення, перехід на /bookings.

TC-BOOK-02. Підтвердження бронювання:

– авторизуватися як власник (ownerId авто);

– відкрити /bookings, вкладка «Запити на оренду»;

– натиснути «Підтвердити».

Очікувано: статус змінюється на confirmed, UI оновлюється.

TC-FAV-01. Додавання/видалення з обраного:

– авторизуватися;

– на картці авто натиснути «серце»;

– перейти на /favorites;

– переконатися, що авто в списку;
– видалити з обраного.

Очікувано: likedCars оновлюється, UI відображає актуальний стан.

3.5 Керівництво користувача (Web-версія)

Для здйснення входу потрібно:
– відкрити сторінку «Увійти»;
– ввести email і пароль або натиснути «Продовжити з Google»;
– після успішного входу користувач перенаправляється на головну сторінку.

Пошук автомобіля:
– на головній сторінці у лівій панелі задати фільтри;
– за потреби вказати дати оренди;
– переглянути список результатів.

Примітка: коли активні фільтри, промо-блоки приховуються, щоб користувач бачив результати без зайвого скролу.

Додавання автомобіля:
– натиснути «Додати авто»;
– додати фото (до 5);
– заповнити характеристики та опис;
– натиснути «Опублікувати».

Бронювання:
– відкрити сторінку авто;
– вибрати дату початку та дату закінчення;
– перевірити «Всього»;
– натиснути «Забронювати».

Обране:
– натиснути «серце» на картці авто;

– перейти в розділ «Обране»;
– переглянути швидкий перегляд або повний список.

ВИСНОВКИ

У межах кваліфікаційної роботи було розроблено web-версію кросплатформеної системи CarShare для peer-to-peer оренди автомобілів. Реалізовано основні бізнес-процеси: пошук та фільтрація авто, створення та керування оголошеннями, створення бронювань та керування їх статусами, ведення обраного та профілі користувачів.

Ключові результати:

– реалізована автентифікація через Firebase Authentication (email/password, Google);
– реалізоване зберігання даних у Firestore з контролем доступу через Security Rules;
– реалізовано зберігання фотографій у форматі base64 без використання Firebase Storage, з обмеженням розміру та кількості фото;
– реалізовано набір сторінок для повного користувацького циклу: каталог, оголошення, бронювання, обране, профілі;
– інтерфейс повністю україномовний, валюта відображається у гривнях.

ПЕРЕЛІК ПОСИЛАНЬ

1. Firebase Documentation : Cloud Firestore. URL: https://firebase.google.com/docs/firestore (дата звернення: 25.04.2025).

2. Firebase Documentation : Authentication. URL: https://firebase.google.com/docs/auth (дата звернення: 29.04.2025).

3. Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 09.05.2025).

4. React Documentation. URL: https://react.dev/ (дата звернення: 21.05.2025).

5. TanStack Query Documentation. URL: https://tanstack.com/query/latest (дата звернення: 07.06.2025).

6. React Hook Form Documentation. URL: https://react-hook-form.com/ (дата звернення: 10.07.2025).

7. Zod Documentation. URL: https://zod.dev/ (дата звернення: 01.08.2025).

8. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (дата звернення: 13.08.2025).

9. React Native Documentation. URL: https://reactnative.dev/docs/getting-started (дата звернення: 24.09.2025).

ДОДАТОК А

Приклад структури документів Firestore

users/{uid}
- name: "..."
- email: "..."
- likedCars: ["carId1", "carId2"]

cars/{carId}
- ownerId: "{uid}"
- images: ["...", ...]
- pricePerDay: 1200
- location: "Київ, Хрещатик"

bookings/{bookingId}
- carId: "{carId}"
- renterId: "{uid}"
- ownerId: "{uid}"
- status: "pending"

Додаток Б. Приклад Firestore Security Rules (скорочений)

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read: if true;
      allow create, update, delete: if request.auth != null && request.auth.uid == userId;
    }
    match /cars/{carId} {
      allow read: if true;
      allow create: if request.auth != null;
      allow update, delete: if request.auth != null && resource.data.ownerId == request.auth.uid;
    }
    match /bookings/{bookingId} {
```

```
    allow read: if request.auth != null &&
      (resource.data.renterId == request.auth.uid || resource.data.ownerId == request.auth.uid);
    allow create: if request.auth != null;
    allow update: if request.auth != null &&
      (resource.data.renterId == request.auth.uid || resource.data.ownerId == request.auth.uid);
    allow delete: if request.auth != null && resource.data.ownerId == request.auth.uid;
   }
  }
}
```