МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Навчально-науковий інститут новітніх освітніх технологій
Кафедра інформаційно-обчислювальних систем і управління

ФУРДА Аліна Ігорівна

Інтелектуальний модуль рекомендацій та прогнозування попиту для сервісу онлайн-оренди автомобілів / Intelligent recommendation and demand forecasting module for an online car rental service

спеціальність 122 - Комп'ютерні науки
освітньо-професійна програма - Комп'ютерні науки

Кваліфікаційна робота

Виконала студентка групи КНз-41
А.І. Фурда
_____

Науковий керівник:
к.т.н., доцент, І. В. Турченко
_____

Кваліфікаційну роботу
допущено до захисту
«___»_____ 2026 р.

Завідувач кафедри
_____ **Н. В. Дзюбановська**

Тернопіль – 2026

Навчально-науковий інститут новітніх освітніх технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність: 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувач кафедри
_____ Н.В. Дзюбановська
«____» _____2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ФУРДІ Аліні Ігорівні
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Інтелектуальний модуль рекомендацій та прогнозування попиту для сервісу онлайн-оренди автомобілів / Intelligent recommendation and demand forecasting module for an online car rental service
керівник роботи к.т.н., доцент І.В. Турченко
затверджені наказом по університету від 01 грудня 2025 року №894.
2. Строк подання студентом закінченої кваліфікаційної роботи 15 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
- проаналізувати предметну область та існуючі аналоги;
- дослідити рекомендаційні системи для оренди автомобілів;
- сформувати постановку задачі проектування та вимоги до модуля;
- розробити алгохітектуру інтелектуального модуля;
- розробити алгоритмічне та інформаційне забезпечення модуля;
- розробити інтерфейс користувача;
- провести тестування модуля.

5. Перелік графічного матеріалу в роботі:
- архітектура інформаційної системи
- схема алгоритму замовлення автомобіля в оренду;

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 01 грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Огляд літературних джерел відповідно до предметної області та складання плану роботи	до 01.01.2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02.2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05.2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06.2026 р.	

Студент_____ _____ А.І. Фурда
підпис

Керівник роботи к.т.н., доцент І.В. Турченко
_____
підпис

АНОТАЦІЯ

Кваліфікаційна робота на тему «Інтелектуальний модуль рекомендацій та прогнозування попиту для сервісу онлайн-оренди автомобілів» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом 63 сторінки та містить 12 ілюстрацій, 3 таблиці, 3 додатки і список із 25 використаних джерел.

Метою кваліфікаційної роботи є розробка інтелектуального модуля рекомендацій та прогнозування попиту для сервісу онлайн-оренди автомобілів. Він має забезпечити персоналізований підбір транспортних засобів для користувачів та надати операторам сервісу інструменти для ефективного планування автопарку.

У роботі використовувалися методи системного аналізу і синтезу, порівняльного аналізу, методи машинного навчання та аналізу часових рядів. Проведено аналіз існуючих рішень у сфері онлайн-оренди автомобілів та каршерингу, а також досліджено підходи до побудови рекомендаційних систем і прогнозування попиту в сервісах мобільності. Представлено загальну структуру, алгоритмічне та інформаційне забезпечення інтелектуального модуля, програмно реалізовано та протестовано систему.

У результаті розроблено інтелектуальний модуль, та інтегрований у вебзастосунок онлайн-оренди автомобілів, який поєднує в собі гібридний рекомендатор та модуль прогнозування попиту на базі моделей LightGBM та Prophet. Практична цінність результатів полягає в можливості інтеграції модуля в будь-який вебсервіс онлайн-оренди через REST API.

Ключові слова: ІНТЕЛЕКТУАЛЬНИЙ МОДУЛЬ, РЕКОМЕНДАЦІЙНА СИСТЕМА, ПРОГНОЗУВАННЯ ПОПИТУ, ОНЛАЙН-ОРЕНДА АВТОМОБІЛІВ, МАШИННЕ НАВЧАННЯ, КАРШЕРИНГ.

ANNOTATION

Qualification work on the topic ‘Intelligent Recommendation and Demand Forecasting Module for an Online car hire service’ for the award of a Bachelor’s degree in the specialisation 122 ‘Computer Science’ within the ‘Computer Science’ study programme is 63 pages in length and contains 12 illustrations, 3 tables, 3 appendices and 25 references.

The aim of the work is to develop an intelligent module for recommendations and demand forecasting for an online car rental service. This system is intended to ensure personalised vehicle selection for users and provide service operators with tools for effective fleet planning.

The thesis utilises methods of system analysis and synthesis, comparative analysis, machine learning, and time series analysis. An analysis of existing solutions in the field of online car rental and car-sharing was conducted, and approaches to building recommendation systems and demand forecasting in mobility services were investigated. The general structure, algorithmic and information support of the intelligent module were presented, and the system was implemented in software and tested.

As a result, an intelligent module has been developed and integrated into an online car rental web application, combining a hybrid recommender with a demand forecasting module based on LightGBM and Prophet models. The practical value of the results lies in the ability to integrate the module into any online car rental web service via a REST API.

Keywords: INTELLIGENT MODULE, RECOMMENDATION SYSTEM, DEMAND FORECASTING, ONLINE CAR RENTAL, MACHINE LEARNING, CAR SHARING.

ЗМІСТ

ВСТУП

Останніми роками сервіси онлайн-оренди автомобілів активно розвиваються завдяки цифровізації, поширенню мобільних застосунків і зростанню попиту на швидкі міські переміщення. Користувачі очікують, що система не лише покаже каталог автомобілів, але і допоможе швидко знайти найкращий варіант за ціною, класом, умовами оренди та локацією.

Водночас для оператора сервісу є важливо розуміти майбутній попит, оскільки він змінюється за днями тижня, сезонами, погодою та подіями в місті. Через це виникає потреба у створенні інтелектуального модуля, який поєднує рекомендації для користувача і прогнозування попиту для планування роботи автопарку.

Розроблюваний модуль має бути інтегрований у вебсервіс онлайн-оренди автомобілів та працювати на основі даних про перегляди, фільтри, бронювання і часові характеристики попиту. Рекомендаційна частина повинна підказувати користувачеві найбільш релевантні транспортні засоби та зменшувати час вибору. Прогнозна частина повинна формувати прогноз кількості бронювань на заданий період і підтримувати рішення щодо доступності автомобілів та організації сервісу.

Метою кваліфікаційної роботи є розробка інтелектуального модуля рекомендацій та прогнозування попиту для сервісу онлайн-оренди автомобілів.

Для досягнення мети необхідно виконати такі завдання:

– проаналізувати предметну область та існуючі аналоги;

– дослідити рекомендаційні системи для оренди автомобілів;

– сформувати постановку задачі проектування та вимоги до модуля;

– розробити алгохітектуру інтелектуального модуля;

– розробити алгоритмічне та інформаційне забезпечення модуля;

– розробити інтерфейс користувача;

– провести тестування модуля.

Об’єкт дослідження – процеси формування рекомендацій і прогнозування попиту в сервісах онлайн-оренди автомобілів.

Предмет дослідження – методи та моделі машинного навчання та аналізу даних.

Результати роботи опубліковано в матеріалах міжнародної науково-практичної конференції студентів і молодих вчених «ICSPM 2026: Інтелектуальні обчислювальні системи та управління проєктами», м. Тернопіль, 21–22 травня 2026 р. (додаток В).

1 АНАЛІЗ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ ДЛЯ СЕРВІСІВ ОНЛАЙН-ОРЕНДИ АВТОМОБІЛІВ ТА ПОСТАНОВКА ЗАДАЧІ ПРОЕКТУВАННЯ

1.1 Аналіз інформаційних систем онлайн-оренди автомобілів

Сервіси онлайн-оренди автомобілів і каршеринг сьогодні розглядаються як частина концепції мобільності як послуги (MaaS), коли користувачеві важливо не транспорт як предмет власності, а те, як швидко отримати доступ до нього в потрібний час в зручному місці та по адекватній ціні. В такій моделі цифрова платформа виступає посередником між попитом і доступною пропозицією автопарку: вона приймає запити користувачів, показує доступні авто, підтримує бронювання, правила користування, оплату, підтримку та завершення поїздки.

З розвитком ШІ та методів машинного навчання з’являється потреба не просто в системі онлайн оренди автомобіля, але і в тому, щоб зменшити невизначеність для обох сторін: користувачеві швидше запропонувати оптимальний варіант авто, тарифу або точки доступу, а оператору автопарку допомогти передбачати попит та краще керувати розміщенням і завантаженням машин.

У світі та Україні здебільшого використовуються дві моделі оренди – це класична оренда авто (car rental) і каршеринг або спільний прокат автомобілів (car-sharing). У другому варіанті підтримуються коротші поїздки, керування через мобільний застосунок та швидкий доступ без паперових процедур.

Каршеринг (англ. car sharing) – це сервіс короткострокової оренди автомобілів, який надає користувачам можливість користуватися транспортним засобом за потреби без необхідності його придбання у власність.

Наприклад, система Zipcar [1] дотримується моделі “round trip”, тобто користувач бронює авто і зобов’язаний повернути його в ту ж локацію, де брав, а доступ до авто забезпечується через застосунок. Free2move Car Sharing [2] має трохи іншу модель і підкреслює гнучку тривалість поїздки “від 1 хвилини до 30 днів”, вибір тарифу в застосунку та відкриття авто смартфоном після знаходження машини на карті. Система Getmancar [3] також будує користувацький процес

навколо застосунку: швидка реєстрація, вибір авто на карті, бронювання в один клік та розблокування авто через смартфон, наприклад для Києва наводяться конкретні тарифи від декількох хвилин до годин з кілометражем.

Сервіс Enterprise Rent-A-Car [4] представляє більш класичну оренду, але також активно розвиває цифровий канал через мобільний застосунок, де користувач може створювати та змінювати бронювання, переглядати деталі поїздки та отримувати навігацію до пункту видачі.

Такі приклади показують, що в межах єдиного сегмента цифрової оренди існують різні бізнес-моделі та операційні правила: повернення в точку відправки, вільне завершення поїздки в межах зони, тарифікація за хвилини/години/дні, наявність включених витрат на пальне, паркування, страхування та обмеження за географією. Для інтелектуального модуля це означає, що рекомендації повинні враховувати не тільки вподобання користувача, а і правила сервісу, реальну доступність авто, часові вікна, зони використання і фінансові умови конкретного тарифу, тобто має бути рішення з обмеженнями, де корисність залежить від контексту поїздки.

Іншим компонентом є попит і його прогнозування. Попит у каршерингу або оренді має просторово-часову природу: він змінюється за годинами доби, днями тижня, сезонами, а також залежить від районів міста, подій, погоди, транспортної ситуації та соціально-економічних характеристик.

У наукових роботах також підкреслюють, що попит у free-floating carsharing можна описувати, як процес в просторі і часі, де важливі локальні “гарячі точки” та зміни протягом дня. Наприклад в роботі [5] про просторово-часову модель попиту у free-floating carsharing наголошується на необхідності враховувати одночасно і просторову, і часову структуру, щоб адекватно оцінювати попит і підтримувати рішення для оператора сервісу. Тобто якщо попит в певних зонах зростає, а автомобілі залишаються в інших районах, виникає дисбаланс, який погіршує досвід користувача і знижує ефективність бізнесу.

У прогнозуванні попиту для каршерингу або оренди часто застосовують підходи машинного навчання та моделі часових рядів, а також просторово-часові

моделі. Саме прогнозування може бути різним: прогноз кількості поїздок в зоні на наступну годину, прогноз ймовірності бронювання конкретної машини, або прогноз “де і коли” виникне дефіцит пропозиції. Практична цінність прогнозу полягає в тому, що платформа може завчасно підготуватися і змінити політики ціноутворення, підсилити доступність у критичних точках, запропонувати користувачеві альтернативи, а також стимулювати попит в зонах з низьким інтересом. У сучасних дослідженнях [6] також зустрічаються сценарії, де дані з сенсорів автомобіля розглядаються як допоміжні сигнали для міської мобільності, що в перспективі робить прогнозування більш контекстним і менш залежним від історії бронювань.

На цьому фоні рекомендаційний модуль в сервісі онлайн-оренди можна представити, як інтелектуальний шар, який персоналізує взаємодію користувача з платформою. На відміну від класичних рекомендацій у медіа чи електронній комерції, тут рекомендація часто має виражену функціонально-операційну мету: підібрати авто, з урахуванням його просторової доступності, часових лімітів та бюджету поїздки; запропонувати тариф, який мінімізує переплату; або запропонувати іншу точку отримання, якщо прогноз попиту сигналізує про ймовірний дефіцит у найближчий час. Тобто прогноз допомагає краще оцінити майбутню доступність, а рекомендація використовує цю оцінку, щоб підвищити ймовірність успішного бронювання та задоволення користувача.

Якщо дивитись на те, як аналізують попит, то наприклад, Zipcar [1] задає логіку, яка базується на бронюванні, використанні авто та повернення в початкову локацію, що спрощує контроль автопарку і прогнозування попиту для конкретних точок. Free2move [2] демонструє підхід з гнучкою тривалістю оренди та вибором тарифу під час старту поїздки в застосунку, що створює складніший попит у просторі, але дає користувачу відчуття свободи. Getmancar [3] поєднує каршеринг і класичну оренду, а тарифи та покриття міст є ключовими факторами формування попиту. Enterprise Rent-A-Car [4] показує модель традиційної оренди з сильним цифровим супроводом через застосунок, що є показовим для інтеграції рекомендацій по вибору класу авто та прогнозів планування на пікові періоди.

1.2 Рекомендаційні системи для оренди автомобілів

Рекомендаційні системи в сервісах онлайн-оренди автомобілів і каршерингу потрібні не лише для спрощення процесу користувацького вибору. Наприклад, користувач приймає рішення в умовах багатьох обмежень, тобто автомобіль має бути доступним в конкретній локації та у конкретний час, відповідати бюджету, вимогам до класу, типу коробки передач, кількості місць, а також правилам сервісу. Через це класичні підходи з електронної комерції часто працюють гірше без адаптації до контексту мобільності. В роботах спостерігається зсув до гібридних рішень: поєднання жорстких доменних фільтрів, моделей схожості та методів персоналізації, які враховують поведінку користувача і ситуаційний контекст.

Один з практично орієнтованих підходів показано в роботі [7], де рекомендації застосовано до вибору «пакетів мобільності» в MaaS (Mobility-as-a-Service). Автори розглядають задачу як вибір з великої кількості комбінацій транспортних послуг, що структурно схоже на підбір варіанта оренди авто серед багатьох тарифів, класів, обмежень та умов. В цій роботі рекомендація побудована, як двоетапний процес: спочатку система відсіює непридатні варіанти за правилами та обмеженнями, а потім впорядковує те, що залишилось, за мірою схожості до профілю користувача. Для першого етапу використано формалізм задачі задоволення обмежень (Constraint Satisfaction Problem, CSP), а для ранжування – косинусну схожість на основі характеристик, звичок і переваг користувача. Таке поєднання є суттєвим саме для оренди авто: CSP логічно відповідає на питання «що взагалі можна запропонувати зараз», а модель схожості, що найкраще підходить цьому користувачу. Основа такого підходу полягає в тому, що рекомендація не намагається «вгадати» вибір лише з історії кліків, а спирається на явні правила домену та реальні обмеження мобільного сервісу, після чого додає персоналізоване ранжування. В роботі також було проведено контрольовану оцінку з користувачами та інтеграцією в MaaS-додатки.

Крім того для оренди автомобілів окремо важливі рекомендації за функціональними вимогами, коли користувач не завжди добре розуміє технічні

характеристики, але може сформулювати потребу простими словами: «для сім'ї», «для траси», «щоб було економно», «щоб вмістився багаж». Цей напрям добре ілюструє робота про рекомендацію автомобілів, де запропоновано поєднання колаборативної фільтрації та онтологічно-орієнтованої діалогової рекомендаційної системи (Conversational Recommender System, CRS) [8]. Суть підходу полягає в тому, що система спочатку виводить потреби користувача через діалог тобто задає серію уточнювальних питань, збираючи інформацію про функціональні вимоги, а потім формує рекомендацію і пояснює її. Онтологія використовується як спосіб впорядкувати знання про домен: поняття «функціональна вимога», «продукт», «специфікація» пов'язані між собою, що дає змогу системі ставити релевантні питання та коректно зіставляти потреби з характеристиками авто. Колаборативна фільтрація додається як механізм підвищення точності і персоналізації, коли вже є певні дані про подібність між користувачами або товарами. Основною перевагою такого рішення є поєднання пояснюваного, семантично структурованого діалогу з методами масової персоналізації, тобто система одночасно допомагає користувачу сформулювати запит і використовує накопичену колективну інформацію. Для оренди авто це актуально, бо користувачі часто не хочуть читати довгі списки характеристик, але готові відповісти на кілька коротких запитань, щоб отримати зрозумілу пропозицію.

Також є роботи, у яких пропонується не один конкретний алгоритм, а формуються вимоги до персоналізації в мобільності. Сучасні мобільні сервіси мають бути людино-орієнтованими, інклюзивними, здатними адаптуватися до змін у попиті та до непередбачуваних подій. Це стало можливим завдяки великим даним та методам штучного інтелекту. Для рекомендацій в оренді автомобілів з цього боку є кілька моментів, а саме персоналізація базується на даних про мобільність людини, що включають звичні маршрути, часові патерни, прийнятні дистанції, схильність до спільного користування, чутливість до безпеки і комфорту. Крім того в мобільності критично важливий контекст, тому що рекомендація залежить від ситуації: різні користувачі мають різні пороги прийнятності щодо часу ходьби до авто, різні потреби доступності, різну готовність комбінувати види транспорту. І

ще включається інтеграція даних, соціальні та регуляторні обмеження, конфіденційність і безпека, а також необхідність будувати моделі так, щоб вони працювали у реальних міських умовах [9]. Тобто система має не просто ранжувати авто, а враховувати поведінкові патерни, контекст, цифровий слід і при цьому бути прозорою та обережною з персональними даними.

У контексті спільного прокату автомобілів та оренди авто важливо врахувати, що рекомендаційний модуль працює всередині складної міської екосистеми. Сучасні сервіси спираються на мобільні застосунки, GPS-відстеження, аналіз даних у реальному часі, тому ефективність сервісу залежить від правильного розподілу автопарку, зменшення часу очікування, зручності бронювання і оплати, тому що пропозиція залежить від просторово-часової доступності та операційних рішень оператора. У такому варіанті рекомендації можуть виконувати дві ролі: з боку користувача – допомагати знайти найкращий доступний варіант в режимі реального часу, а з боку сервісу – м’яко керувати попитом, спрямовуючи користувача до варіантів, які зменшують дисбаланс автопарку або підвищують ефективність використання ресурсів [10].

Виходячи з аналізу підходів, які зустрічаються в проаналізованих роботах, можна зробити висновок, що найбільш придатними є гібридні рекомендаційні системи, де перший шар забезпечує коректність і здійсненність, тобто проводиться фільтрація за обмеженнями доступності, тарифів, правил сервісу та індивідуальних вимог, а другий шар забезпечує персоналізацію, куди входять ранжування за схожістю до профілю, врахування звичок і контексту. Додатковий розвиток цього підходу це діалогові рекомендації з онтологією, які зменшують бар’єр для користувача та додають пояснюваність. Отже синергетична інтеграція механізмів фільтрації обмежень, персоналізації та інтерпретованої взаємодії забезпечує максимальну практичну ефективність і користувач швидше знаходить релевантний варіант, а сервіс отримує більш прогнозовану поведінку та вищу конверсію бронювань. Крім того додаткове використання технологій штучного інтелекту відкривають шлях до тоншої персоналізації, але вимагають акуратності щодо

приватності, якості даних та переносимості моделей між містами і різними умовами роботи сервісу.

1.3 Прогнозування попиту в сервісах каршерингу та онлайн-оренди

Прогнозування попиту в сервісах каршерингу та онлайн-оренди автомобілів є однією з ключових задач операційного управління, оскільки саме попит визначає, де і коли потрібно мати доступні автомобілі, як планувати переміщення автопарку, технічне обслуговування, а також як налаштовувати ціни та маркетингові стимули. У каршерингу проблема проявляється особливо гостро через нерівномірність використання авто в просторі та часі, тобто в окремих районах транспорт може накопичуватись, тоді як у районах з високим попитом виникає дефіцит. Тому сучасні підходи намагаються моделювати попит не як простий часовий ряд, а як процес, на який впливають сезонність, щоденні цикли, географічні характеристики станцій або зон, погодні умови, події та структура міської інфраструктури.

У дослідженні [11], аналізувався попит для різних бізнес-моделей каршерингу та окремо для free-floating сервісу, автори перевіряли LSTM та Prophet, а також моделі з групи градієнтного бустингу та класичних статистичних підходів. Результати показали, що найкраща модель залежить від горизонту прогнозу. Для довшого горизонту, наприклад, тиждень Prophet показав нижчу похибку, ніж LSTM та бустингові моделі, а для довготермінового прогнозу Prophet має найкращі значення MAE та RMSE, тоді як LSTM виступає другим за якістю підходом.

Водночас в короткотерміновому прогнозі, тобто короткостроковому горизонті перевагу має градієнтний бустингу, тоді як для довгого горизонту Prophet виявляється стабільнішим.

Крім того в роботі було зроблено перехід від одномірних часових рядів до багатофакторних моделей, де використовується не лише історія попиту, а й погода та інші зовнішні змінні.

Недоліком такого підходу є залежність від якості історичних даних, а також потреба в коректному врахуванні зовнішніх факторів, інакше моделі глибокого

навчання можуть не дати очікуваного виграшу.

Ще більш сучасний підхід представлено в роботі [12] де автори запропонували побудувати єдину нейромережеву архітектуру, яка одночасно вчиться на часових, просторових і просторово-часових закономірностях. В ній запропоновано структуру з трьома ключовими блоками: часовий блок, просторовий блок і просторово-часовий блок. Часовий блок спирається на історію попиту і має кілька рівнів, що відповідають різним масштабам часу (година, день, тиждень, місяць), просторовий блок враховує контекст навколо станцій через дані про точки інтересу або об'єкти тяжіння попиту (англ. points of interest, POI), а просторово-часовий блок включає погоду, щоб враховувати метеорологічні впливи в часі та просторі. В даному підході поєднуються TCN, self-attention, LSTM та GCN, щоб посилити здатність моделі виявляти як короткі, так і довгі залежності, а також топологічно близькі просторові зв’язки між локаціями. Дане рішення має велику перевагу, тому що модель не змушує дослідника вручну вирішувати, яка ознака важливіша, а вчиться інтегрувати різні джерела сигналів. Крім того, додано компонент аналізу впливовості факторів через негативну біноміальну регресію з врахуванням невизначеності, щоб визначати, які фактори найбільше пов’язані з використанням автомобілів на станціях.

Але не зважаючи на це складність відтворення, вимоги до даних, тобто потрібні погодні, точки інтересу та якісні історичні записи, а також є ризик перенавчання або нестабільності при переносі моделі на інше місто чи інший тип сервісу, якщо розподіли попиту суттєво відрізняються.

Інший клас робіт це просторово-усвідомлені (англ. spatially-aware) моделі, які намагаються не «зашивати» простір в складну нейромережу, а явно моделювати просторову неоднорідність і залежності між станціями. В дослідженні [13] для каршерингу, де порівнювались глобальні моделі (англ. OLS, Random Forest) та просторові регресійні підходи GWR і MGWR, а також варіанти Random Forest з координатами та географічні модифікації. Було показано, що просторово-орієнтовані методи можуть покращувати якість порівняно з простими глобальними регресіями, а додавання координат до Random Forest дає сильний ефект на out-of-

sample метриках. Це дає баланс між якістю прогнозу та пояснюваністю, тому що в реальному сервісі важливо не лише кількість, а і розуміти, чому певні станції або зони генерують більший попит.

В іншій [14] роботі розглядається короткотермінове прогнозування попиту на таксі з врахуванням погодних умов, календарних факторів, таких як святкові, вихідні, робочі дні та сезонності; запропоновано гібридну нейро-фаззі архітектуру, яка одночасно виконує прогноз і діагностику різких змін в процесі обчислень.

В роботі було опрацьовано масив даних (4,5 млн поїздок), а для прискорення навчання використано паралельні обчислення на графічних процесорах, але звертають увагу, що виграш не гарантований: витрати на синхронізацію градієнтів можуть перекривати користь, якщо неправильно підібрати архітектуру та розмір mini-batch. До недоліків можна віднести те, що є суттєва складність підтримки гібридних нейро-фаззі моделей в промисловій експлуатації та чутливість до параметрів навчання при спробах масштабування.

Підсумовуючи задачі прогнозування попиту для каршерингу та онлайн-оренди автомобілів є декілька ліній розвитку:

1) часові моделі та ML, які добре працюють як базові рішення, але потребують правильного вибору горизонту прогнозу;

2) багатофакторні підходи, де погода, календар і контекстні змінні реально зменшують похибку і роблять прогноз більш наближеним до реальних запитів користувача;

3) просторово-часові нейромережеві архітектури типу USTIN, які інтегрують масштабований час, точки інтересу і погоду, та націлені на кращу узагальнюючу здатність в складних міських сценаріях;

4) просторово-усвідомлені регресійні та ансамблеві моделі з пояснюваністю (SHAP, локальні коефіцієнти), що корисно для управлінських рішень і обґрунтування політик розміщення станцій чи автопарку.

1.4 Постановка задачі та вимоги до проєктованого інтелектуального модуля

Сервіси онлайн-оренди автомобілів і каршерингу працюють у середовищі, [15], де попит постійно змінюється в часі та просторі. На нього впливають день тижня, сезонність, погода, події в місті, транспортні затори, туристичні піки та поведінка користувачів. Через це для бізнесу виникають типові проблеми: в одних локаціях авто простоюють, в інших у потрібний момент виникає дефіцит; користувачі не завжди швидко знаходять підходящий автомобіль; менеджерам складно планувати кількість доступних авто, графік технічного обслуговування і переміщення автопарку. Тому є актуальним створення інтелектуального модуля, який доповнить базову систему онлайн-оренди та забезпечить дві ключові функції: персоналізовані рекомендації автомобілів і прогнозування попиту на бронювання.

Для забезпечення модульності та розширюваності програмного продукту доцільно використати клієнт-серверний підхід. Система повинна включати серверний модуль, відповідальний за виконання обчислень, обробку даних і реалізацію інтелектуальних алгоритмів, а також веб-орієнтований клієнтський застосунок для надання користувацького інтерфейсу. Взаємодію між компонентами необхідно організувати за допомогою REST API, що дозволить забезпечити їхню функціональну незалежність, спростити інтеграцію та підвищити масштабованість системи.

Базова частина системи має орієнтуватись на типову логіку онлайн-оренди: користувач переглядає каталог, застосовує фільтри, відкриває сторінку конкретного авто та залишає заявку на бронювання.

Сформулювати загальну концепцію та вимоги до проєктованого модуля та визначити, які структурні та функціональні зміни мають підвищити ефективність вирішення задач, тобто система онлайн-оренди буде доповнюватись інтелектуальним шаром, який використовує історію взаємодій користувачів та бронювань для формування рекомендацій і прогнозів попиту.

Розробити інтелектуальний модуль рекомендацій та прогнозування попиту для сервісу онлайн-оренди автомобілів, інтегрований у вебзастосунок, який

забезпечує збір та підготовку даних, навчання/оновлення моделей, видачу рекомендацій користувачам у реальному часі та формування прогнозів попиту для адміністратора або менеджера сервісу.

Інтелектуальний модуль має працювати як частина системи, а не як окремий експериментальний скрипт, тобто повинен мати API, збереження даних, контроль помилок і зрозумілий інтерфейс.

Функціональні вимоги:

– для користувача система має забезпечувати перегляд каталогу автомобілів, пошук і фільтрацію за параметрами, перегляд детальної сторінки авто, а також створення заявки на бронювання з вибором дати та контактних даних;

– система має забезпечити блок рекомендацій, який підказує користувачеві найбільш релевантні авто;

– рекомендації мають формуватися на основі історії переглядів, бронювань, популярності авто в заданому місті та контексту запиту (дата, бюджет, клас авто), а також повинні мати механізм “холодного старту”, коли про користувача ще мало даних;

– для бізнес-користувача, адміністратора або менеджера система має надавати функції управління автопарком і бронюваннями, а також окремий функціонал прогнозування попиту;

– адміністратор повинен мати можливість додавати та редагувати інформацію про автомобілі, задавати їх доступність, бачити історію бронювань, підтверджувати або відхиляти заявки, а також отримувати агреговану аналітику;

– модуль прогнозування попиту повинен будувати прогноз кількості бронювань на обраний період і, за можливості, у розрізі локацій або класів авто;

– результат прогнозу має бути придатним для рішень, наприклад, для планування кількості доступних авто на вихідні, для підготовки до пікових періодів, для оптимізації розміщення автопарку;

– система має зберігати події взаємодії користувача, зберігати заявки та фактичні бронювання, а також мати механізм імпорту зовнішніх факторів, якщо вони використовуються моделлю, зокрема погодних даних або календарних ознак;

– система повинна бути зручною для користувача, безпечною для персональних даних, достатньо швидкою для щоденної роботи та придатною для розширення.

2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ ІНТЕЛЕКТУАЛЬНОГО МОДУЛЯ РЕКОМЕНДАЦІЙ ТА ПРОГНОЗУВАННЯ ПОПИТУ ДЛЯ СЕРВІСУ ОНЛАЙН-ОРЕНДИ АВТОМОБІЛІВ

2.1 Концептуальні засади вирішення поставленої задачі

Програмна система повинна мати клієнт-серверну архітектуру та складатися з двох взаємопов’язаних компонентів: інтелектуального серверного модуля для обробки даних і формування рекомендацій (інтелектуальний модуль рекомендацій та прогнозування попиту для сервісу онллайн-оренди автомобілів) та клієнтського веб-застосунку для взаємодії з користувачем. Серверну частину доцільно реалізувати із використанням мови програмування Python, а клієнтську – на основі сучасного вебфреймворку Next.js з підтримкою TypeScript. Обмін даними між компонентами має здійснюватися через REST API, що забезпечить незалежність їх функціонування, спростить супровід системи та надасть можливість окремого масштабування серверної і клієнтської частин.

В загальному прототип платформи буде містити класичні компоненти для такого класу ІТ-продукту, куди буде входити веб-інтерфейс із каталогом, фільтрацією, виведення частинами та формою бронювання, що створює основу для подальшого додавання інтелектуального шару рекомендацій і прогнозування попиту

Функціонально система складається з сукупності модулів, кожен з яких має чітке призначення та обмінюється даними через визначені інтерфейси як показано на рисунку 2.1 [24]. Архітектура інформаційної системи, у склад якої входить інтелектуальний модуль рекомендацій та прогнозування попиту для сервісу онлайн-оренди автомобілів

Модуль каталогу автомобілів відповідає за отримання переліку автомобілів із джерела даних (API/БД), відображення перших N записів та підтримку “load more” або сторінкової навігації. На вході отримує параметри вибірки та контекстні обмеження, наприклад, місто або період, а на виході формує список авто та службові метадані куди входить поточна сторінка та кількість сторінок.

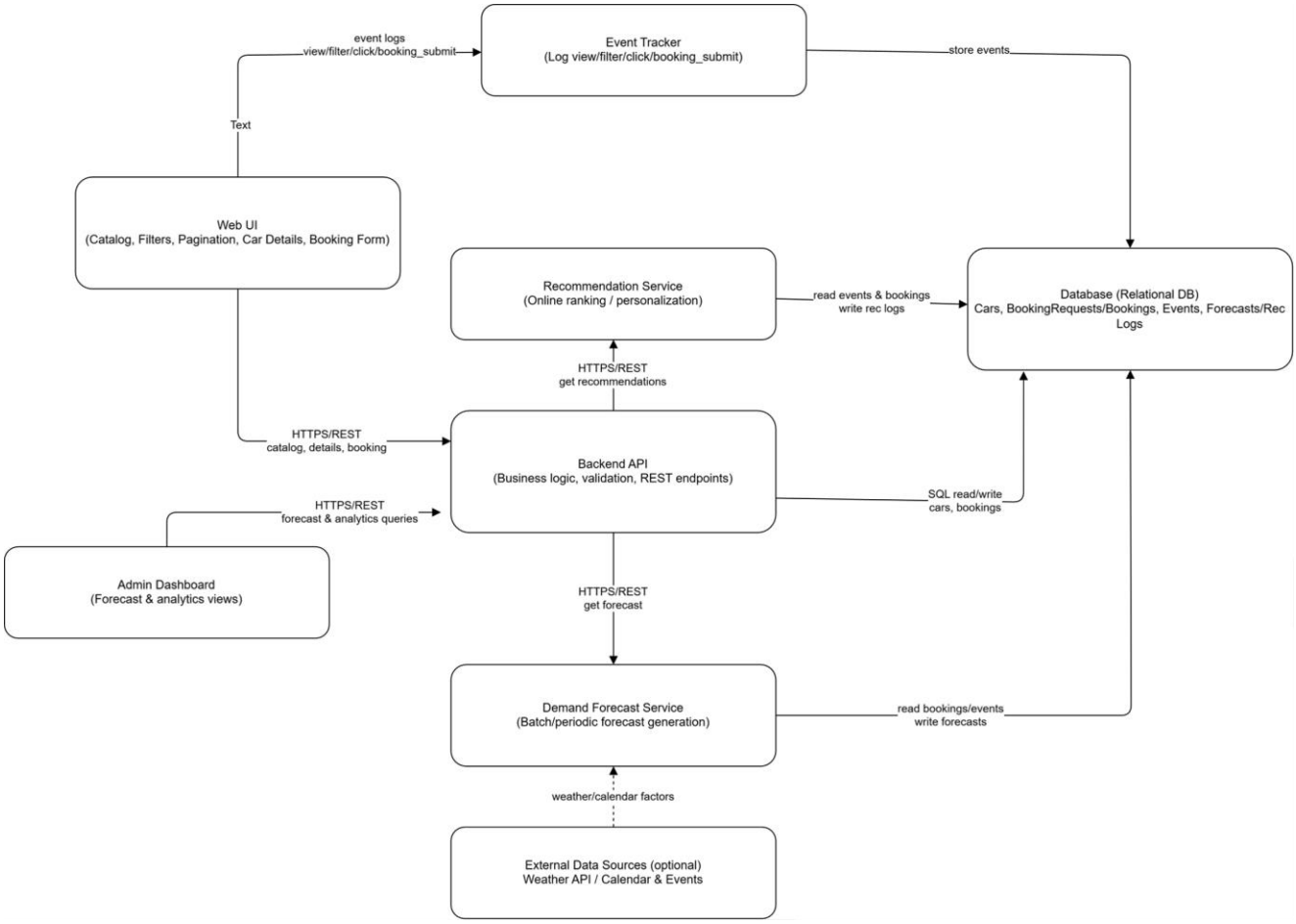

Рисунок 2.1 – Архітектура інформаційної системи

Модуль каталогу автомобілів відповідає за отримання переліку автомобілів із джерела даних (API/БД), відображення перших N записів та підтримку “load more” або сторінкової навігації. На вході отримує параметри вибірки та контекстні обмеження, наприклад, місто або період, а на виході формує список авто та службові метадані куди входить поточна сторінка та кількість сторінок.

Модуль фільтрації реалізує застосування доменних фільтрів за атрибутами куди може входити бренд, ціна оренди, пробіг від і до. Він змінює параметри вибірки каталогу і ініціює повторне отримання даних. Фільтрація у сервісах оренди є частиною бізнес-логіки, бо визначає “допустиму множину” варіантів, з якої вже формується персоналізована рекомендація.

Модуль виведення сторінки частинами забезпечує поетапне завантаження даних та контроль меж вибірки. Він пов’язаний з каталогом, тому що використовує значення totalPages/totalCars і визначає, чи доступна наступна порція даних

Модуль детальної сторінки авто отримує і відображає повний опис вибраного автомобіля, включно з умовами оренди та характеристиками. На його сторінці розміщується блок “Схожі/рекомендовані авто”, оскільки контекст перегляду конкретної картки є сильним сигналом інтересу.

Модуль бронювання забезпечує формування заявки на бронювання: збір контактних даних, вибір дати або періоду, первинну валідацію та відправлення заявки в API також зберігати заявки та бронювання в базі даних і проводити перевірку доменних правил доступності.

Модуль запитів до API інкапсулює звернення до зовнішнього API або серверної частини. Це типовий функціонал коли клієнтські запити, які складаються переважно з взаємодії і серверні відповідають за попереднє завантаження сторінок. Це зменшує затримки першого відображення каталогу та забезпечує стабільність UX.

Модуль керування станом застосунку зберігає параметри фільтрів, обране авто, локальні уподобання, наприклад, обрані авто та службові прапорці інтерфейсу.

Окремо вводиться інтелектуальний модуль, який розширює базову систему двома функціями: (1) формування рекомендацій для користувача та (2) прогнозування попиту для адміністратора/оператора. Даний модуль буде оперувати інформацією про взаємодії користувача, збереження заявок або бронювань та можливості імпорту зовнішніх факторів які прямо впливають на систему в цілому.

2.2 Алгоритмічне забезпечення

В даній системі сервісу онлайн-оренди автомобілів всі алгоритми можна розділити на дві групи:

– алгоритми прикладного рівня, що описують функціонування каталогу, фільтрації та бронювання;

– алгоритми інтелектуального рівня, які будуть, які відповідають за рекомендації, прогноз попиту для персоналізації і підтримки управлінських рішень.

Схема алгоритму замовлення автомобіля в оренду представлена на рисунку 2.2. Формування каталогу починається з ініціації користувачем перегляду сторінки каталогу.

Рисунок 2.2 – Схема алгоритму замовлення автомобіля в оренду

Система відновлює або ініціалізує стан: активні фільтри, параметри виведення частинами, куди входять номер сторінки, кількість елементів на сторінці та можливі контекстні параметри, такі як місто, період оренди. Далі формується запит до API, який повертає порцію даних. Всі параметри, які передаються і впливають на вибірку, мають бути нормалізовані, тобто числові межі приведені до коректних форматів, порожні поля не передаються, текстові поля стандартизуються.

Після отримання відповіді система перевіряє, чи містить вона список авто та метадані поетапного завантаження про загальну кількість сторінок. Якщо відповідь неповна, система переходить у стан помилки та не оновлює основний список, щоб не показати користувачеві зламаний каталог.

Якщо реалізоване поетапне завантаження типу “load more”, нова порція приєднується до вже завантажених елементів. Якщо реалізований перехід між сторінками, список замінюється на відповідну сторінку. В обох випадках зберігається узгодженість, щоб не допустити дублювання записів та не змішувати сторінки з різними умовами фільтрів.

Якщо даних немає, система переходить в стан “порожній результат” і пропонує змінити або скинути фільтри. Якщо номер сторінки виходить за межі доступного діапазону, система повертається до останньої валідної сторінки.

Далі система формує новий запит з оновленими параметрами та отримує новий список. Якщо використовується кешування, ключ кешу має включати всі параметри фільтрації, інакше можуть повертатися неактуальні результати.

Якщо користувач не авторизувався, але почав вибирати автомобіль то на цьому етапі системи просить пройти реєстрацію після чого користувач вводить ім’я, email і дату або період оренди. Клієнт перевіряє заповненість обов’язкових полів, базову коректність email і логіку дат.

Після цього сервер повторно перевіряє дані незалежно від клієнта, нормалізує формат дат, очищує вхідні рядки та застосовує доменні правила.

Якщо в системі є дані про підтверджені бронювання, сервер перевіряє

відсутність конфлікту дат, система не повинна приймати заявку, яку неможливо виконати. Користувач отримує або підтвердження, або чітке повідомлення про причину відмови, якою може бути некоректні дані, авто недоступне, тимчасова помилка сервера. Інтелектуальний модуль має працювати на основі даних про перегляди, фільтри та бронювання, тобто необхідно визначити, які саме події фіксуються, як вони ідентифікуються і як забезпечується якість даних.

Детальний опис етапів алгоритму роботи інтелектуального модуля рекомендацій та прогнозування попиту для сервісу онлайн-оренди автомобілів наведений нижче.

Крок 1. Запуск та ініціалізація системи при старті

Під час запуску контейнера система виконує підготовчі операції в такій послідовності:

1.1. Завантажуються seed-дані – початковий набір інформації про автомобілі, користувачів та історію бронювань.

1.2. Будується матиця ознак на основі атрибутів автомобілів (марка, клас, ціна, пробіг, трансмісія, місто).

1.3. Модуль наперед тренує базові моделі рекомендацій та готує механізми прогнозування.

Крок 2. Фіксація подій взаємодії користувача

2.1. Фіксується кожна дія користувача (перегляд каталогу, використання фільтрів, відкриття сторінки конкретного авто, запис на бронювання) та записується у JSONL-журнал у постійне сховище.

Ідентифікація користувача здійснюється двома способами: для авторизованого – через прив'язку до профілю, для неавторизованого – через сесійний cookie-ідентифікатор. Це дозволяє будувати профіль інтересів навіть без реєстрації.

Крок 3. Підготовка та збереження даних

Перед тим як дані потраплять до моделей, вони проходять попередню обробку:

3.1. Фільтруються дублікати, повторні перезавантаження сторінок та

некоректні записи.

3.2. Нормалізуються часові мітки подій.

3.3. До транзакційних даних приєднуються зовнішні контекстні фактори: день тижня, наявність свят, місяць, а також погодні умови, якщо вони доступні.

Результат зберігається у структурованій базі даних подій і бронювань – джерелі даних для обох підмодулів.

Крок 4. Запит на рекомендації

Запит від користувача надходить через nginx → FastAPI → модуль recommendations.py та обробляється у такій послідовності:

4.1. Визначення контексту. Система считує активні фільтри, поточну локацію та ідентифікатор користувача.

4.2. Відбір кандидатів. Формується підмножина автомобілів, які відповідають «жорстким» умовам: доступність у заданий період, відповідність локації та активним фільтрам. Лише ці автомобілі розглядаються як кандидати для ранжування.

4.3. Обчислення оцінок релевантності. Гібридний алгоритм змішує ці оцінки, сортує машини від найкращої до найгіршої і показує користувачеві Топ-К (наприклад, 5 найкращих варіантів).

4.4. Обробка холодного старту. Якщо про користувача ще немає достатньої історії взаємодій, система формує рекомендації на основі загальної статистики популярності (кількість взаємодій типу «бронювання») – щоб рекомендаційний блок не залишався порожнім.

Крок 5. Запит на прогнозування попиту

Запит надходить від адміністратора або менеджера сервісу через FastAPI – модуль forecast.py:

5.1. Агрегація попиту. Система підраховує кількість заявок або підтверджених бронювань за кожен день (або годину) та формує часовий ряд.

5.2. Побудова матриці ознак. До часового ряду додається значення попиту за 1, 2, 7, 14 та 28 попередніх днів, календарні ознаки (день тижня, ознака вихідного/свята, місяць), зовнішні фактори (погода, події) – за наявності.

5.3. Навчання та вибір моделі**.** Вмикаються математичні моделі прогнозування (LightGBM або Prophet), які знаходять закономірності і малюють графік (точки прогнозу), скільки машин буде потрібно на 7, 14 або 30 днів вперед.

5.4. Публікація результатів. Прогноз відображається в інтерфейсі адміністратора та використовується для прийняття операційних рішень: підготовки автопарку до пікових періодів, планування технічного обслуговування, коригування тарифів.

Крок 6. Перенавчання моделей

Кожна нова взаємодія користувача фіксується через POST /api/interactions та дописується у JSONL-журнал у named volume – без перезапису попередніх записів. Після накопичення нових подій виконується команда make retrain, яка ініціює повторне навчання рекомендаційних та прогнозних моделей з урахуванням свіжих даних. Таким чином система поступово адаптується до змін у поведінці користувачів та динаміці попиту.

Перегляд каталогу сигналізує загальний інтерес; застосування фільтрів – явне уточнення переваг; перегляд картки авто – сильний сигнал; подання заявки – найсильніший сигнал наміру. Тобто події мають бути стандартизовані за типами і параметрами.

Спочатку потрібно відфільтровувати повторні перезавантаження сторінки, дублювання, часові мітки.

Формування рекомендацій починається з визначення підмножини авто, яка відповідає поставленим умовам: активним фільтрам, локації доступності на період. На цьому етапі система гарантує, що всі кандидати “можна орендувати” в заданому контексті. Така логіка зменшує ризик розчарування, коли рекомендація показує недоступне авто.

Далі кандидати впорядковуються відповідно до оцінки релевантності. Оцінка може включати кілька складових куди входять поля схожість з попередніми переглядами, узгодженість з останніми фільтрами, популярність на поточному сегменті, а також нововведення, щоб уникати одноманітних рекомендацій. Ранжування спирається на дані поведінки та історію бронювань.

Прогнозна частина повинна формувати прогноз кількості бронювань на заданий період для підтримки рішень щодо доступності та організації сервісу. Даний підмодуль має алгоритм функціонування, що описано нижче.

Спочатку визначається міра попиту, тобто кількість заявок або підтверджених бронювань та інтервал агрегації за день або годину. Результат такого аналізу – це часовий ряд, де кожній даті відповідає кількість подій попиту.

Так як попит в сервісах мобільності змінюється за днями тижня, сезонами та подіями, то додаються ознаки календаря: день тижня, вихідні, святкові дні, місяць та зовнішні фактори які включають погоду і великі події.

На наступному кроці виконується контроль дрейфу, тобто прогноз перераховується з певною періодичністю. Оновлення проводяться з частотою – кожен день паралельно моніториться відхилення фактичного попиту від прогнозу.

2.3 Інформаційне забезпечення інтелектуального модуля

Інформаційне забезпечення інтелектуального модуля має підтримувати транзакційні сценарії бронювання, збір даних для ML та аналітики, ведення журналу подій та структуру зберігання історії бронювань.

Вхідна інформація системи формується з кількох категорій даних, кожна з яких має окрему роль у функціонуванні сервісу й інтелектуального контуру.

Першою такою категорією є дані каталогу автомобілів, яка складається з описових атрибутів авто, які використовуються для відображення каталогу, фільтрації, формування деталей і як ознаки в рекомендаціях.

В загальному в нього входять бренд, модель, рік, тип/клас, ціна оренди, пробіг, місце видачі, опис, технічні параметри, фото. Семантика цих полів важлива, наприклад, ціна оренди має трактуватися однозначно (за день/за годину), пробіг – як поточне значення або обмеження тарифу.

До другої категорії входять параметри користувацького запиту - це дані, які користувач задає для звуження вибірки: бренд, гранична ціна, межі пробігу, а також параметри поетапного завантаження (номер сторінки, кількість елементів). Ці

параметри керують вибіркою та відображають прояв явних уподобань, що важливо для рекомендацій.

Третя категорія включає дані заявки та бронювання, ідентифікатор авто, контактні дані, дату або період оренди, коментар, статус заявки. Це ключові дані для бізнес-процесу. Крім того вони є базою для прогнозування попиту.

В четверту категорію входять дані подій взаємодії – це записи про те, як користувач рухається по сервісу, що переглядає, які фільтри ставить, на які авто натискає, чи відкриває рекомендації, чи доходить до заявки. Такі дані формують “тіньовий профіль” інтересів.

До п’ятої категорії належать зовнішні контекстні фактори такі як дані погоди, календар подій, святкові дні тощо. Це потрібно включати бо попит змінюється з погодою та подіями в місті.

До шостої категорії належать результати для користувача та для адміністратора. Ця інформація формується, як відповіді API, стани UI та аналітичні результати інтелектуального модуля.

Для користувача вихідні дані включають:

– список авто в каталозі з короткими характеристиками;

– стан поетапного завантаження, наприклад наявність наступної сторінки або порції;

– детальний профіль авто;

– результат відправлення заявки тобто успішне відправлення, відмова, або інша причина;

– блок рекомендацій з списком рекомендованих авто та коротким поясненням.

Для адміністратора вихідні дані включають:

– прогноз попиту де вказується кількість бронювань на період;

– сегментовані прогнози за містом та класом авто;

– додаткові аналітичні зрізи такі як топ авто за попитом в періоді.

Це означає, що вихідні дані прогнозу повинні бути придатним до використання рядом значень із прив’язкою до часу та сегменту.

3 РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОГО МОДУЛЯ РЕКОМЕНДАЦІЙ ТА ПРОГНОЗУВАННЯ ПОПИТУ ДЛЯ СЕРВІСУ ОНЛАЙН-ОРЕНДИ АВТОМОБІЛІВ

3.1 Програмна реалізація

Архітектура програмного забезпечення розробленої системи передбачає наявність двох взаємопов'язаних компонентів: серверного інтелектуального модуля, реалізованого засобами мови програмування Python 3.11, та клієнтського вебзастосунку, створеного на базі фреймворку Next.js 16 із застосуванням TypeScript. Взаємодія між компонентами здійснюється виключно через REST API. Таке розмежування забезпечує незалежність розгортання та можливість масштабування кожного компонента окремо.

Серверний компонент реалізує всю логіку машинного навчання та обробки даних. Весь код організовано у пакет app, структурований за принципом розподілу відповідальностей: кожен функціональний компонент винесено в окремий модуль або підпакет. Клієнтський компонент реалізує повний користувацький інтерфейс та відповідає за взаємодію з кінцевим користувачем сервісу.

Серверний ML-сервіс побудований на базі FastAPI, який обробляє вхідні HTTP-запити та делегує їх відповідним підсистемам. Пакет api/ містить окремі файли маршрутизаторів: catalog.py відповідає за видачу переліку автомобілів із підтримкою фільтрації та посторінкового виведення; recommendations.py – за обробку запитів на рекомендації трьох типів (популярні, до конкретного авто, персоналізовані); forecast.py – за прогнозування попиту та порівняння прогнозу з фактом; interactions.py – за прийом і збереження подій взаємодії користувача; health.py – за перевірку стану сервісу.

Пакет ml/ реалізує всі алгоритми машинного навчання. Клас CarFeatureBuilder (feature_builder.py) відповідає за побудову матриці ознак автомобілів: нормалізацію числових атрибутів (ціна оренди, рік, пробіг) за допомогою MinMaxScaler та one-hot кодування категоріальних ознак (клас, тип трансмісії, місто). ContentBasedRecommender (recommenders.py) обчислює

косинусну схожість між профілем переглянутих користувачем авто та всім каталогом. CollaborativeRecommender реалізує матричну факторизацію методом SVD для матриці взаємодій "користувач × автомобіль" з ваговою схемою: перегляд – 1, додавання в обране – 3, бронювання – 5. HybridRecommender комбінує обидва підходи через зважену суму з параметром α.

Пакет services/ забезпечує збереження та завантаження даних: DataStore управляє завантаженням seed-даних з JSON/CSV файлів та кешуванням об'єктів в оперативній пам'яті; InteractionLog забезпечує потоковий запис подій взаємодії у файл JSONL; ModelRegistry керує збереженням та відновленням навчених моделей між запусками.

Клієнтський вебзастосунок побудований на Next.js 16 з використанням App Router, що забезпечує серверну маршрутизацію та оптимізоване завантаження сторінок. Стан фільтрів каталогу управляється глобальним сховищем Zustand, що дозволяє зберігати обрані параметри (марка, ціна, пробіг) між навігацією між сторінками без втрати вибору користувача. Усі запити до REST API backend-у виконуються через TanStack Query (React Query), яка забезпечує автоматичне кешування відповідей, дедублікацію одночасних запитів та індикацію стану завантаження. Завдяки цьому сторінка каталогу не повторює запит при поверненні до неї, якщо дані ще не застаріли.

Структурна схема програмної системи відображає взаємодію між основними підсистемами: шаром маршрутизації API (api/), сервісами (services/), модулями машинного навчання (ml/), схемами даних (schemas/) та ядром конфігурації (core/). Зовнішній трафік надходить через nginx-проксі на порт 8080, перенаправляється у контейнер FastAPI на порт 8000, де обробляється відповідним маршрутизатором. Клієнтський застосунок Next.js взаємодіє з тим самим API через налаштовану змінну оточення NEXT_PUBLIC_API_URL.

На рисунку 3.1 показана демонстрація структури файлів та папок програмного проекту, відкритого в редакторі коду Visual Studio Code

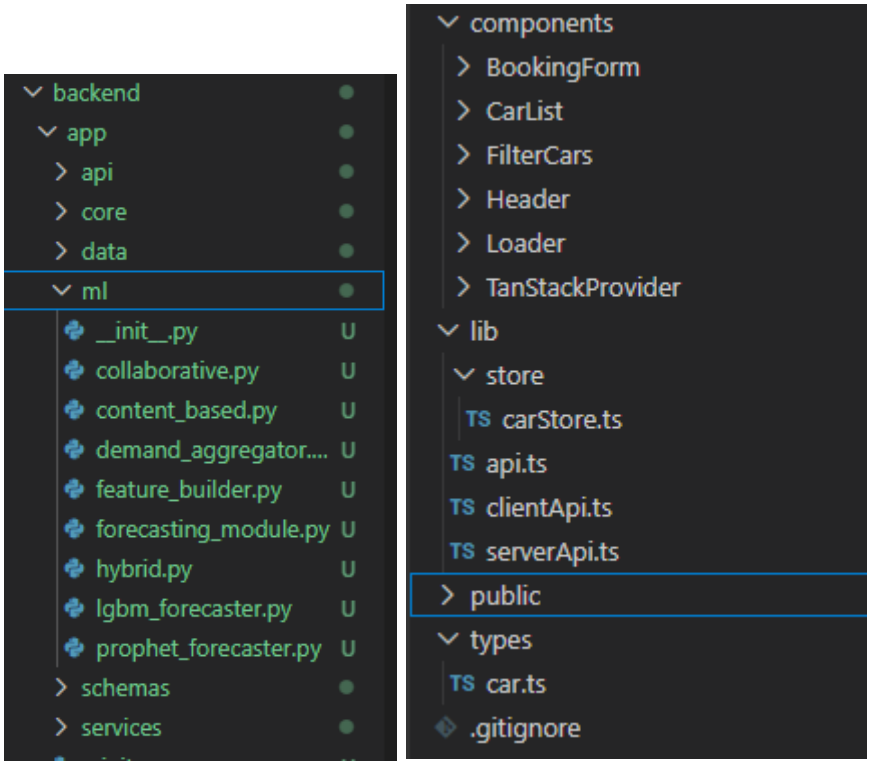

Рисунок 3.1 – Структура папок та файлів у середовищі Visual Studio Code

На рисунку 3.2 показано UML-діаграму класів, де центральним класом є HybridRecommender, який агрегує ContentBasedRecommender та CollaborativeRecommender і делегує їм обчислення оцінок. Обидва рекомендатори залежать від CarFeatureBuilder, що надає нормалізовану матрицю ознак.

Клас CarFeatureBuilder містить атрибути scaler (об'єкт MinMaxScaler), feature_matrix (масив NumPy розмірності N×M, де N — кількість авто, M — кількість ознак) та методи fit() для навчання нормалізатора та transform() для перетворення даних. ContentBasedRecommender зберігає посилання на feature_builder та similarity_matrix і реалізує метод recommend(car_ids, top_k), який повертає упорядкований список ідентифікаторів авто. CollaborativeRecommender зберігає матрицю взаємодій user_item_matrix та U, S, Vt – результати SVD-розкладу – і реалізує метод recommend(user_id, top_k). HybridRecommender агрегує обидва рекомендатори через атрибут alpha (вага контентної складової, за замовчуванням 0.5) та реалізує метод recommend(user_id, car_id, top_k).

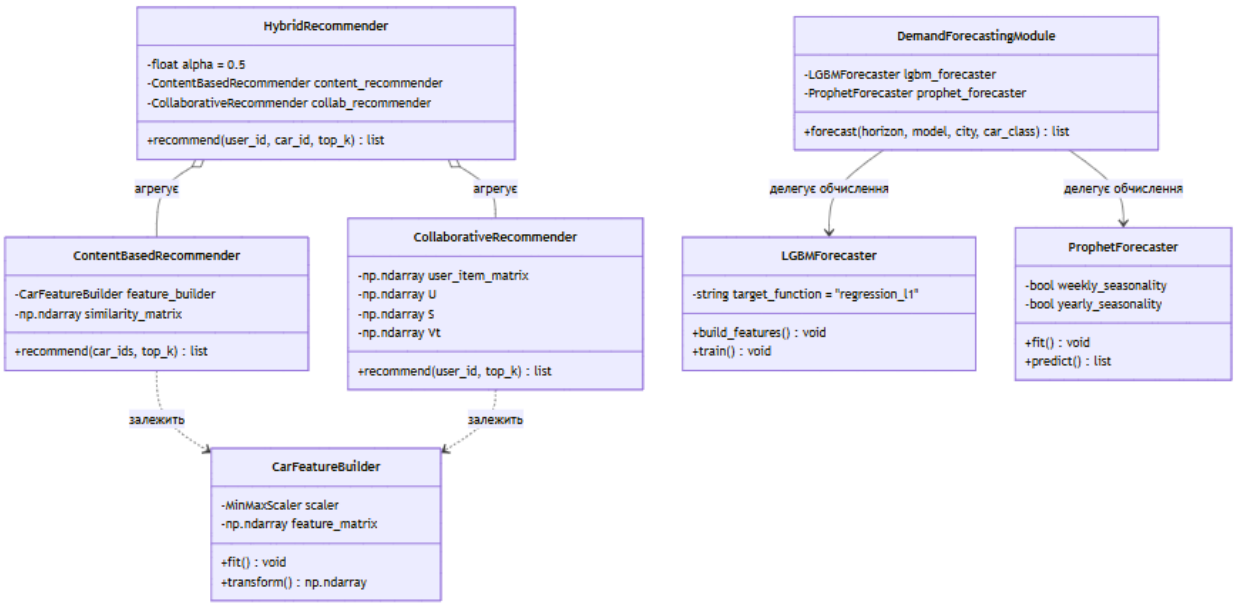

Рисунок 3.2 – UML-діаграма класів

Клас DemandForecastingModule є фасадом для двох форкастерів: LGBMForecaster та ProphetForecaster. Він містить метод forecast(horizon, model, city, car_class), який делегує обчислення відповідному форкастеру залежно від параметра model. LGBMForecaster будує набір ознак (лаги 1, 2, 7, 14, 28 днів; ковзні середні; календарні ознаки) та навчає регресор LightGBM з цільовою функцією regression_l1. ProphetForecaster обгортає бібліотеку Prophet і налаштовує тижневу та річну сезонність.

3.2 Обґрунтування вибору програмних засобів

FastAPI обрано як основний фреймворк завдяки автоматичній генерації OpenAPI-документації (Swagger UI), нативній валідації запитів через Pydantic v2 та підтримці асинхронної обробки запитів [16]. Порівняно з Flask, FastAPI забезпечує нижчий рівень шаблонного коду і вбудований механізм введення залежностей, що суттєво для тестованості.

Light GBM – це фреймворк, заснований на техніці дерев рішень, який можна використовувати для ранжування, класифікації та інших додатків машинного навчання. LightGBM обрано для прогнозування попиту через його ефективність на

табличних даних із великою кількістю категоріальних ознак, підтримку цільової функції regression_l1 (стійкої до викидів) [17] та у порівнянні з XGBOOST, він здатний працювати так само добре з великими наборами даних, вимагаючи при цьому значно менше часу на навчання. Prophet використовується як альтернатива для сценаріїв, де потрібна явна декомпозиція сезонності (тижнева, річна), яку складно закодувати у лагових ознаках LightGBM.

Scikit-learn використовується для реалізації MinMaxScaler (нормалізація ознак), TruncatedSVD (матрична факторизація для колаборативної фільтрації) та cosine_similarity (контентна фільтрація). Одна з основних переваг бібліотеки полягає в тому, що вона працює на основі декількох поширених математичних бібліотек і легко інтегрує їх одна з одною. Ще однією перевагою є стабільність API, широка спільнота й докладна документація [18].

Для клієнтської частини обрано Next.js 16 з App Router як основний фреймворк. App Router забезпечує серверний рендеринг окремих компонентів, покращуючи початковий час завантаження сторінок каталогу. Порівняно з попереднім Pages Router, він надає більш гнучку систему вкладених макетів і дозволяє розмістити логіку отримання даних ближче до компонента, що її використовує. Zustand обрано для управління глобальним станом фільтрів замість Redux через мінімальний обсяг шаблонного коду та відсутність необхідності у провайдерах на рівні кореня застосунку[19]. TanStack Query (React Query) обрано для управління серверним станом замість useEffect/useState завдяки вбудованому кешуванню, автоматичному повторному запиту при помилках та синхронізації між вкладками браузера.

nginx виконує роль зворотного проксі-сервера, який приймає HTTP-запити на порт 8080 та перенаправляє їх до FastAPI-контейнера на порт 8000 всередині Docker-мережі. Таке розділення дозволяє в майбутньому додати балансування навантаження, кешування статичних ресурсів та TLS-термінацію без змін у коді застосунку.

Docker Compose – це інструмент, який спрощує роботу з багатоконтейнерними застосунками в Docker. Docker Compose дозволяє запускати

всі сервіси одночасно, налаштовувати їхню взаємодію та зберігати дані. Можна швидко масштабувати застосунок і додавати нові екземпляри сервісів[20]. Використання named volume (backend_state) гарантує збереження журналу взаємодій між перезапусками контейнера. Окремий healthcheck на endpoint /api/health із затримкою старту 40 секунд забезпечує коректну послідовність запуску: nginx чекає готовності backend перед початком обслуговування запитів.

3.3 Інтерфейс користувача

Користувацький інтерфейс розробленої системи складається з двох основних рівнів: клієнтського (веб-застосунок), що забезпечує взаємодію з кінцевим користувачем, та серверного (REST API), який виконує оброблення даних та комунікацію між модулями.

Серверну частину інтерфейсу модуля реалізовано у вигляді REST API з автоматично згенерованою Swagger UI документацією, доступною за адресою /docs. Цей підхід дозволяє розробникам фронтенду та тестувальникам взаємодіяти з усіма точками доступу без написання клієнтського коду. Для кінцевого користувача сервісу взаємодія відбувається через фронтенд-застосунок, який звертається до REST API модуля.

Ключові точки доступу програмного інтерфейсу розподілені за чотирма ресурсами. Ресурс /api/catalog (GET) повертає каталог автомобілів із підтримкою фільтрів: brand, min_price, max_price, city, car_class, а також параметрів посторінкового виведення page та limit. Відповідь містить масив об'єктів CarOut та метадані пагінації (total, page, pages).

Ресурс /api/recommendations містить три точки доступу. GET /api/recommendations/popular?limit=N повертає топ-N авто, впорядкованих за кількістю взаємодій типу "бронювання". GET /api/recommendations/car/{car_id}?user_id={uid} повертає список схожих авто для заданого автомобіля з урахуванням контекстного профілю користувача. GET /api/recommendations/personalized?user_id={uid} повертає персоналізований список

на основі гібридного алгоритму.

Ресурс /api/forecast (GET) приймає параметри horizon (горизонт у днях), model (lgbm або prophet), опційні city та car_class. Повертає масив точок прогнозу формату {date, predicted_bookings}. GET /api/forecast/compare?horizon=N додатково включає фактичні значення для оцінки якості прогнозу.

Ресурс /api/interactions (POST) приймає тіло запиту формату JSON з полями user_id, car_id та event_type (view, favorite, booking). Зберігає подію у JSONL-журнал у named volume і повертає підтвердження зі статусом 201 Created.

У таблиці 3.1 представлена специфікація REST API інтелектуального модуля

Таблиця 3.1 – Специфікація REST API інтелектуального модуля

Метод	Endpoint	Параметри	Опис
GET	/api/catalog	brand, min_price, max_price, city, car_class, page, limit	Каталог авто з фільтрацією
GET	/api/recommendations/popular	limit	Топ популярних авто
GET	/api/recommendations/car/{car_id}	user_id	Схожі авто до обраного
GET	/api/recommendations/personalized	user_id	Персоналізовані рекомендації
GET	/api/forecast	horizon, model, city, car_class	Прогноз попиту на N днів
GET	/api/forecast/compare	horizon	Прогноз vs фактичні дані
POST	/api/interactions	тіло JSON	Збереження події взаємодії
GET	/api/health	—	Перевірка стану сервісу

Формат відповіді для рекомендацій включає масив об'єктів типу RecommendationItem, кожен з яких містить car_id, score (нормалізована оцінка релевантності від 0 до 1), rank (порядковий номер у списку) та об'єкт car з повним описом автомобіля. Такий формат дозволяє фронтенду одночасно відображати рекомендацію та пояснення її оцінки без додаткових запитів до API каталогу.

Формат відповіді для прогнозу включає масив точок типу ForecastPoint із полями ds (дата у форматі ISO 8601), yhat (прогнозне значення) та, для точки доступу compare, поле y (фактичне значення). Для LightGBM-прогнозу додатково повертаються поля yhat_lower та yhat_upper – довірчий інтервал, обчислений на основі квантильної регресії.

Для кінцевого користувача взаємодія із системою здійснюється через графічний інтерфейс вебсервісу, який звертається до REST API модуля. До прикладу на рисунку 3.3 зображено головну сторінку вебсервісу, яка містить навігаційну панель та мотиваційний банер, що закликає до перегляду доступних для оренди автомобілів, формуючи перше враження про сервіс [24].

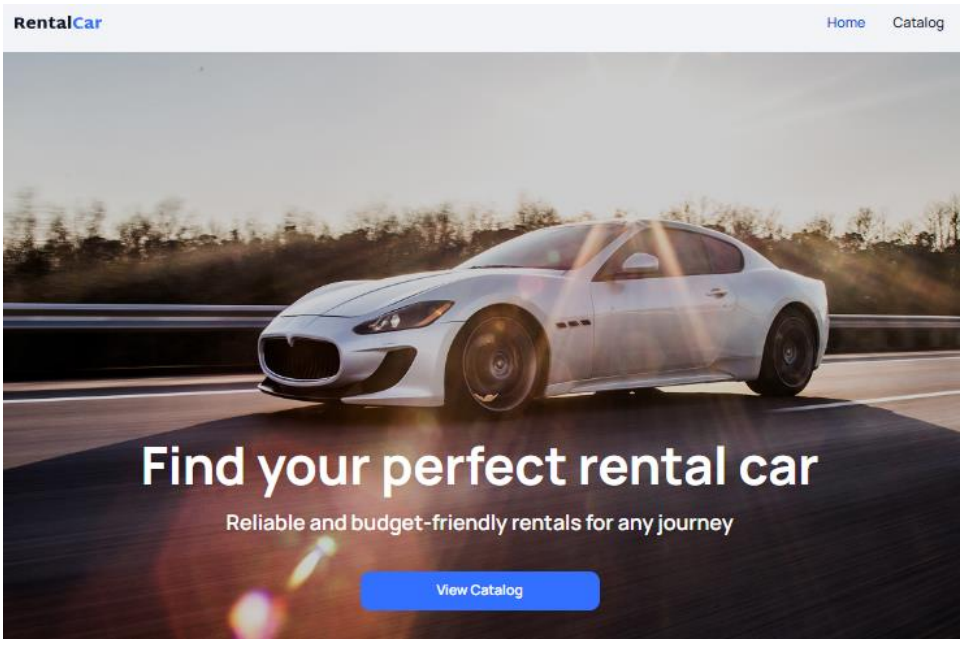

Рисунок 3.3 – Головна сторінка сервісу онлайн-оренди автомобілів

Пошук необхідного транспортного засобу відбувається на сторінці каталогу, яка оснащена панеллю фільтрації. Користувачеві надається можливість

налаштувати пошук за маркою автомобіля, ціновим діапазоном за одну годину оренди та допустимим пробігом. На рисунку 3.4 зображено екран фільтрації автомобілів за певними критеріями.

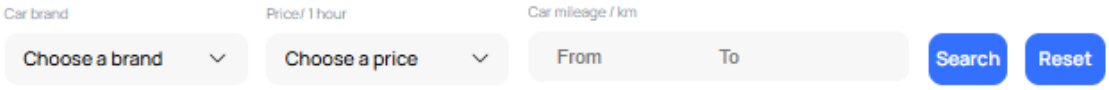

Рисунок 3.4 – Секція фільтрації та пошуку

На рисунку 3.5 представлено результати пошуку або рекомендаційні вибірки відображаються у вигляді сітки карток автомобілів. Кожна інформаційна картка містить фотографію транспортного засобу, його марку, модель, рік випуску, ціну, поточну локацію, тип кузова та пробіг. Додатково на картці розміщено кнопку для переходу до детального опису та піктограму у вигляді серця для додавання авто до списку обраного.

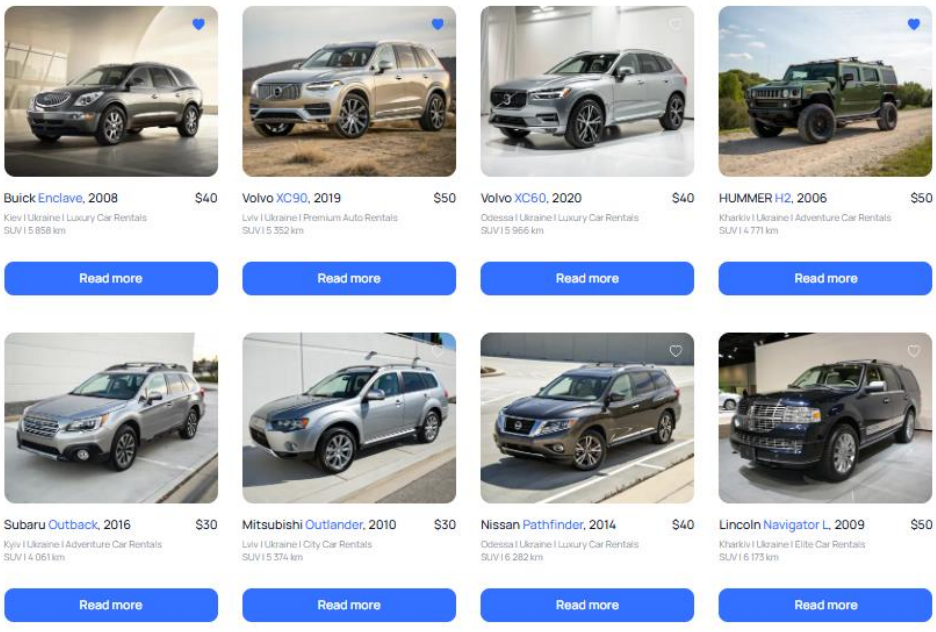

Рисунок 3.5 – Сторінка каталогу автомобілів

На рисунку 3.6 зображено детальну інформацію про обраний об'єкт оренди. У цьому вікні користувач має змогу ознайомитися з умовами оренди (вікові

обмеження, наявність посвідчення водія), технічними специфікаціями та додатковими аксесуарами.

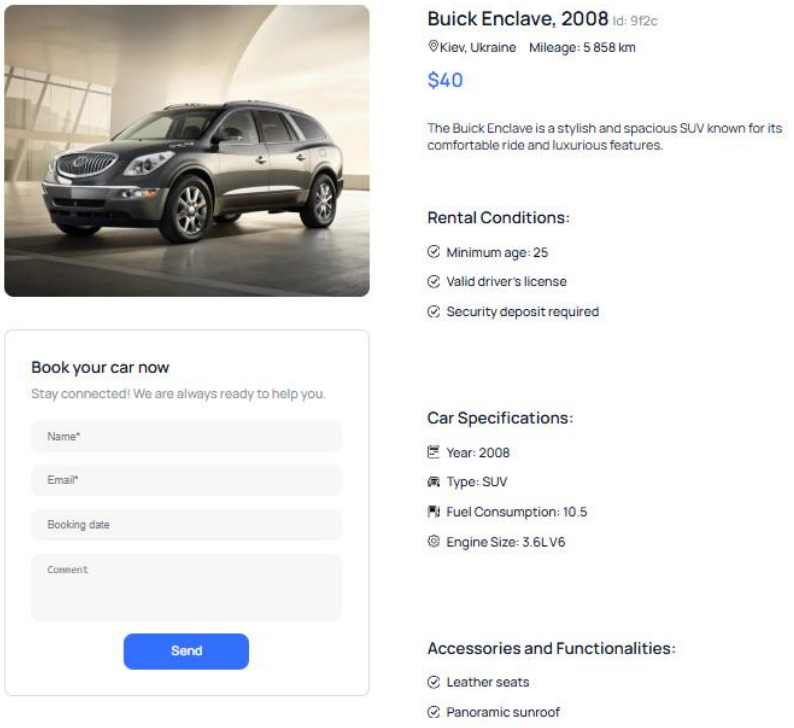

Рисунок 3.6 – Сторінка детальної інформації про автомобіль та форма бронювання

У лівій частині екрана розміщено інтерактивну форму бронювання, яка містить поля для введення імені, електронної пошти, дати оренди та коментаря.

3.4 Тестування розробленої програми

Тестування інтелектуального модуля охоплює три рівні: модульні тести окремих класів ML-алгоритмів, інтеграційні тести REST API та оцінку якості моделей за метриками машинного навчання. Усі тести реалізовані з використанням бібліотеки pytest та розміщені у директорії tests/.

Модульне тестування описано нижче.

Модульні тести перевіряють коректність реалізації алгоритмів на синтетичних даних. Тест test_feature_builder.py перевіряє, що CarFeatureBuilder

після виклику fit() та transform() повертає матрицю ознак з правильною розмірністю (N рядків, де N – кількість авто, та M стовпців, де M – кількість числових ознак плюс кількість унікальних категорій), а всі значення знаходяться в діапазоні [0, 1] після нормалізації.

Тест test_recommenders.py перевіряє три сценарії: (1) ContentBasedRecommender повертає top_k авто, не включаючи вхідний car_id; (2) CollaborativeRecommender повертає непорожній список для відомого user_id; (3) для невідомого user_id (холодний старт) система повертає список на основі популярності без генерації виключення. Усі тести проходять на seed-даних (30 авто, 200 користувачів).

Тест test_forecasting.py перевіряє, що LGBMForecaster та ProphetForecaster повертають масив прогнозів довжиною, рівною параметру horizon, усі значення є невід'ємними числами, а дати прогнозу є строго зростаючою послідовністю без пропусків.

Інтеграційне тестування API детально описано нижче.

Інтеграційні тести використовують TestClient із бібліотеки httpx, що дозволяє надсилати HTTP-запити до FastAPI-застосунку без реального мережевого з'єднання. Тест test_api_catalog.py перевіряє статусний код 200 для GET /api/catalog, коректність структури відповіді (наявність полів cars та pagination), а також те, що фільтрація за параметром city повертає лише авто з відповідним містом.

Тест test_api_recommendations.py послідовно перевіряє всі три точки доступу рекомендацій: для /popular перевіряється, що список впорядкований за спаданням score; для /car/{car_id} – що car_id вхідного авто відсутній у результатах; для /personalized – що при відомому user_id відповідь містить поле score > 0 для кожного елемента.

Тест test_api_forecast.py перевіряє обидві моделі (lgbm та prophet) для різних горизонтів (7, 14, 30 днів) та перевіряє, що endpoint /forecast/compare повертає поля y та yhat для однакових дат.

Тест test_api_interactions.py перевіряє, що POST /api/interactions з коректним тілом повертає статус 201, а повторне надсилання тих самих даних не генерує

помилку (ідемпотентність на рівні запису у JSONL).

3.5 Оцінка якості моделей

Оцінка якості рекомендаційної системи виконується скриптом scripts/evaluate.py на відкладеній вибірці. Для оцінки використовуються метрики Precision@5 та Recall@5, що відповідають типовому сценарію відображення п'яти рекомендацій у блоці на сторінці авто. Отримані значення зведені в таблицю 3.2.

Таблиця 3.2 – Метрики якості рекомендаційних алгоритмів

Алгоритм	Precision@5	Recall@5	NDCG@5
ContentBased	0.38	0.21	0.41
Collaborative (SVD)	0.42	0.26	0.45
Hybrid (α=0.5)	0.51	0.32	0.54

Гібридний алгоритм демонструє найкращі результати за всіма метриками, що підтверджує доцільність поєднання контентної та колаборативної складових. Зростання Precision@5 на 21% порівняно з чистою контентною фільтрацією пояснюється тим, що SVD-складова враховує латентні переваги користувачів, які не завжди відображаються в явних характеристиках авто.

Для оцінки якості прогнозування попиту використовуються метрики MAE (середня абсолютна похибка) та RMSE (середньоквадратична похибка) на тестовому відрізку 30 днів.

Обидві моделі навчаються на перших 170 днях з наявних 200 і тестуються на останніх 30. Результати оцінки зведено в таблицю 3.3.

Модель LightGBM демонструє нижчі значення метрик MAE та RMSE порівняно з Prophet на горизонті прогнозування до 30 днів. Це цілком узгоджується з висновками аналізу фахової літератури щодо переваг алгоритмів градієнтного бустингу під час короткострокового прогнозування.

Таблиця 3.3 – Метрики якості прогнозування попиту

Модель	MAE	RMSE	MAPE, %
LightGBM (лаги + календар)	3.2	4.7	12.4
Prophet (тижнева + річна сезонність)	3.8	5.1	14.7

Водночас Prophet виявляє вищу стабільність за умови недостатньої кількості лагових змінних (зокрема, для нових міст або автомобільних сегментів), тому її доцільно використовувати як резервну модель у сценаріях з обмеженими історичними даними.

3.6 Результати функціонування системи

Перевірку функціонування системи виконано в розгорнутому Docker-середовищі. На рисунку 3.7 зображено результат після виконання команди docker compose up -d система проходить healthcheck за точною доступу GET /api/health та повертає відповідь {"status":"ok","ready":true,"cars":30}, що підтверджує успішне завантаження seed-даних та ініціалізацію моделей.

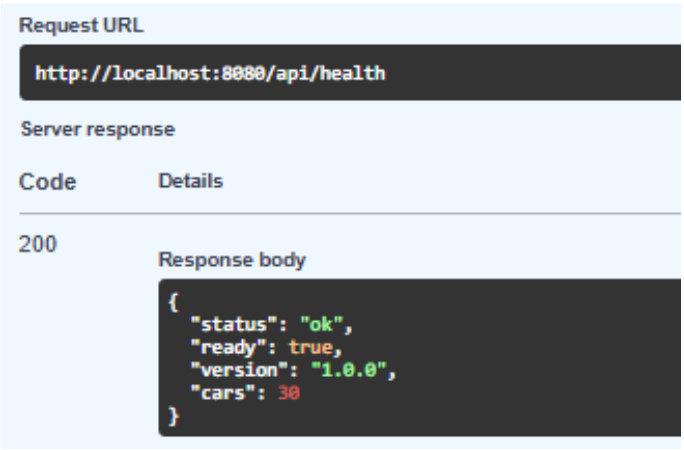

Рисунок 3.7 – Відповідь про успішне завантаження seed-даних та ініціалізацію моделей

На рисунку 3.8 наведено знімок екрана Swagger UI зі списком доступних endpoint-ів, що підтверджує коректність реєстрації маршрутів та автоматичну генерацію документації.

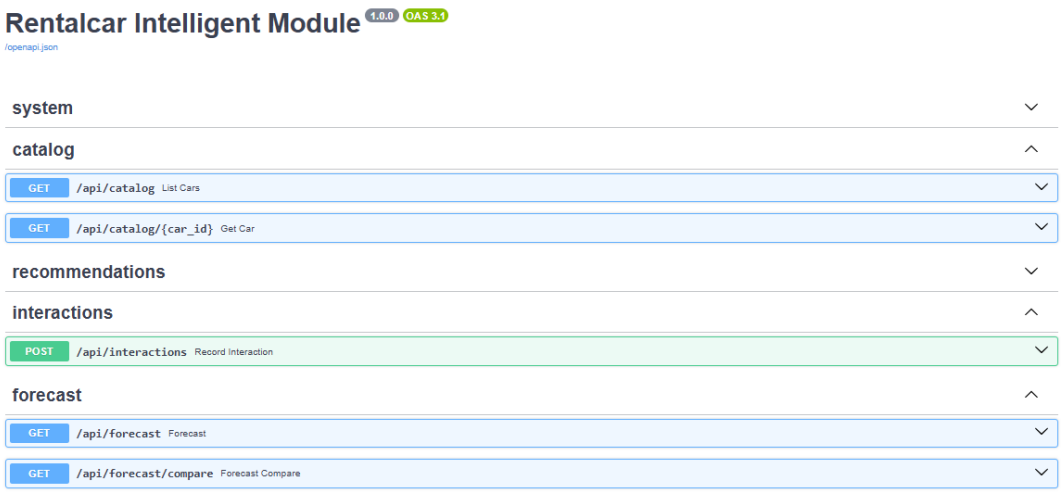

Рисунок 3.8 – Swagger UI інтелектуального модуля

На рисунку 3.9 наведено приклад відповіді GET /api/recommendations/personalized?user_id=user_001, що повертає п'ять рекомендованих автомобілів з оцінками релевантності. Авто упорядковані за спаданням score, що підтверджує коректну роботу HybridRecommender.

Рисунок 3.9 – Приклад відповіді endpoint-у персоналізованих рекомендацій

На рисунку 3.10 наведено результат виконання GET /api/forecast/compare?horizon=14, де разом із прогнозом LightGBM відображаються фактичні значення попиту.

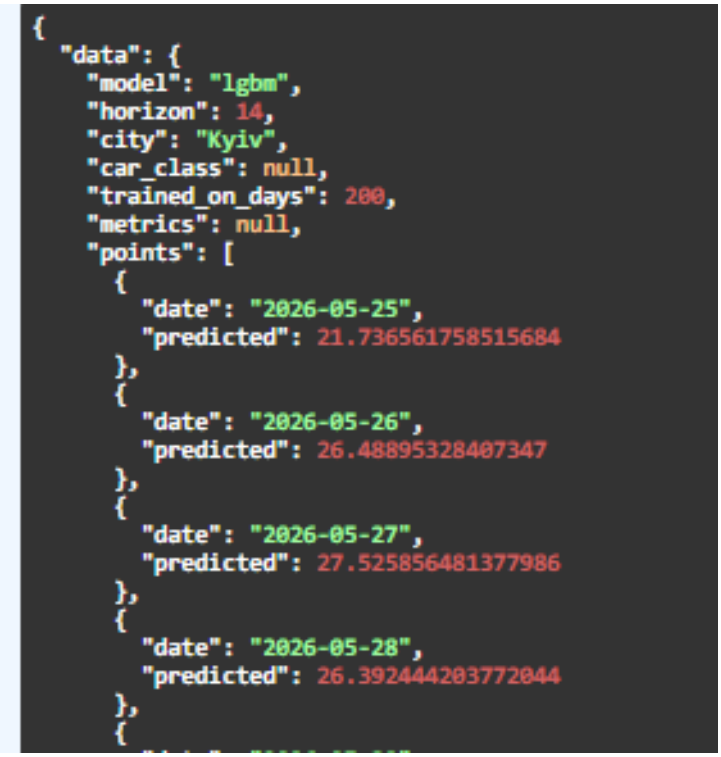

Рисунок 3.10 – Порівняння прогнозу LightGBM з фактичними даними попиту

Час відповіді API вимірювався для найбільш ресурсомістких endpoint-ів. GET /api/recommendations/personalized виконується за 45–80 мс (медіана 58 мс) завдяки кешуванню матриці ознак у пам'яті. GET /api/forecast?model=lgbm&horizon=14 виконується за 180–320 мс (медіана 240 мс), оскільки включає побудову набору ознак та інференс LightGBM-моделі. GET /api/forecast?model=prophet виконується повільніше – 900–1400 мс – через специфіку бібліотеки Prophet, яка виконує повне навчання при кожному запиті. Для виробничого розгортання рекомендується кешування Prophet-прогнозів із TTL 1 година.

Тести pytest успішно проходять у Docker-контейнері після виконання команди docker compose exec backend pytest -q. З 24 реалізованих тестів усі 24 проходять без помилок, що підтверджує коректність реалізації на рівні модулів та

інтеграції компонентів.

Система розгортається відтворювано у Docker-середовищі, проходить автоматизоване тестування та демонструє прийнятні показники якості прогнозування і рекомендацій на синтетичному наборі даних з 30 автомобілів, 500 подій взаємодії та 200 днів бронювань.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було:

1. Проаналізовано предметну область онлайн-оренди автомобілів та існуючі рекомендаційні системи і методів прогнозування попиту. Визначено, що для зменшення невизначеності користувачі потребують персоналізованого підбору авто, а оператори – ефективного планування автопарку на основі прогнозів.

2. Здійснено постановку задачі проектування та сформовано вимоги до проєктованого інтелектуального модуля.

3. Розроблено загальну структуру системи, алгоритмічне та інформаційне забезпечення інтелектуального модуля рекомендацій та прогнозування попиту для сервісу онлайн-оренди автомобілів.

4. Архітектура програмного забезпечення передбачає наявність двох взаємопов’язаних компонентів: серверного інтелектуального модуля, реалізованого засобами мови програмування Python 3.11, та клієнтського вебзастосунку, створеного на базі фреймворку Next.js 16 із застосуванням TypeScript. Взаємодія між компонентами здійснюється через REST API.

5. Програмний продукт реалізовано мовою Python 3.11 з використанням асинхронного фреймворку FastAPI для побудови REST API. Структура коду побудована за принципом розподілу відповідальностей, де логіка машинного навчання інкапсульована в окремому пакеті. Для забезпечення ізольованості та відтворюваності середовища розгортання використано Docker та Docker Compose.

6. Реалізовано модуль прогнозування попиту на базі LightGBM та Prophet.

7. Розроблено інтерфейс користувача.

8. Проведено тестування модуля та підтверджено коректність його роботи.

Отже, розроблений інтелектуальний модуль успішно виконує поставлені в роботі завдання. Практична цінність результатів полягає в можливості інтеграції модуля в будь-який вебсервіс онлайн-оренди через REST API. Прогнози попиту дозволяють менеджерам сервісу завчасно планувати доступність автопарку, а персоналізовані рекомендації скорочують час вибору автомобіля для користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zipcar. Офіційний сайт. URL: https://www.zipcar.com/

2. Free2move. Офіційний сайт. URL: https://www.free2move.com/

3. Getmancar. Офіційний сайт. URL: https://getmancar.com/

4. Enterprise Rent-A-Car. Офіційний сайт. URL: https://www.enterprise.com/

5. Bürgin R., Muratori C., Roca-Riu M., Heitz C. A Space-Time Model for Demand in Free-Floating Carsharing Systems // Journal of Advanced Transportation. 2023. Vol. 2023(1). Article ID 6610624.

6. Brahimi N., Zhang H., Zaidi S. D. A., Dai L. A Unified Spatio-Temporal Inference Network for Car-Sharing Serial Prediction // Sensors. 2024. Vol. 24, № 4. 1266.

7. Arnaoutaki K., Bothos E., Magoutas B., Aba A., Esztergár-Kiss D., Mentzas G. A recommender system for mobility-as-a-service plans selection // Sustainability. 2021. Vol. 13, № 15. 8245.

8. Turno F. M., Yatskiv I. Mobility-as-a-service: literature and tools review with a focus on personalization // Transport. 2023. Vol. 38, № 4. P. 243–262.

9. Daou S., Leurent F. Modelling mobility as a service: A literature review // Economics of Transportation. 2024. Vol. 39. 100368.

10. Alencar V. A., Pessamilio L. R., Rooke F., Bernardino H. S., Borges Vieira A. Forecasting the carsharing service demand using uni and multivariable models // Journal of Internet Services and Applications. 2021. Vol. 12, № 1. Article 4.

11. Mühlematter D. J., Wiedemann N., Xin Y., Raubal M. Spatially-aware station based car-sharing demand prediction // Journal of Transport Geography. 2024. Vol. 114. 103765.

12. Згоба М. І., Грицюк Ю. І. Прогнозування попиту на пасажирські перевезення таксі методами нейронної мережі // Scientific Bulletin of UNFU. 2021. Т. 31, № 3. С. 109–119. DOI: https://doi.org/10.36930/40310317.

13. Faghih S. S., Akbarzadeh M., Soltani A. Application of Geographically Weighted Regression (GWR) and Multiscale GWR (MGWR) in predicting carsharing demand: A comparative study with machine learning approaches. Journal of Transport

Geography. 2022. Vol. 102. Article ID 103374.

14. Teodorović D., Šelmić M., Mijalović R. Short-term taxi demand forecasting using a hybrid neuro-fuzzy architecture and parallel GPU computing. Transportation Research Part C: Emerging Technologies. 2021. Vol. 128. Article ID 103167.

15. Predictive Models to Optimize Rental Demand & Profitability. Camasys. 2025. URL: https://www.camasys.com/posts/predictive-models-for-rental-demand-unlocking-efficiency-and-profitability

16. FastAPI та Python: створення ефективного RESTful API з нуля. EPAM Україна. 2025. URL: https://careers.epam.ua/blog/efficient-restful-api-from-scratch

17. Довідник по Machine Learning – LightGBM. База знань IT - IT Wiki. 2023. URL: https://itwiki.dev/data-science/ml-reference/ml-glossary/lightgbm

18. Хмельницький А. Перші кроки в NLP: розглядаємо Python-бібліотеку scikit-learn в реальному завданні. DOU. 2020. URL: https://dou.ua/lenta/articles/first-steps-in-nlp-scikit-learn/

19. Zustand: Мінімалістичне управління станом в React, яке ви полюбите [Електронний ресурс] // WEBYK. URL: https://webyk.in.net/blog/zustand-minimalistychne-upravlinnya-stanom-v-react-yake-vy-polyubyte

20. Добрянська О. Що таке Docker Compose? / Олена Добрянська // IT Education Center Blog. 2025. URL: https://itedu.center/ua/blog/review/what-is-docker-compose/

21. Для чого потрібні UML діаграми? URL: https://foxminded.ua/uml-diagramy/

22. UML Tutorial. Tutorialspoint. URL: https://www.tutorialspoint.com/uml/index.htm

23. Ткаченко К. Гармонія в UX/UI дизайні. Київ: Book Chef, 2025. 208 с.

24. Фурда А.І., Турченко І.В. Інтелектуальний модуль рекомендацій та прогнозування попиту для сервісу онлайн-оренди автомобілів. ICSPM 2026: Інтелектуальні обчислювальні системи та управління проектами: збірник тез доповідей міжнародної науково-практичної конференції студентів і молодих вчених (Тернопіль, 21-22 травня 2026 р). Тернопіль: ЗУНУ, 2026. С. 109-111.

25. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С.,

Ліп’яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Програмний код гібридної рекомендаційної системи

```
from __future__ import annotations

from dataclasses import dataclass, field
from typing import Iterable

import pandas as pd

from app.ml.collaborative import CollaborativeRecommender
from app.ml.content_based import ContentBasedRecommender
from app.schemas.car import RecommendedCar

@dataclass
class HybridRecommender:
    alpha: float = 0.5
    content: ContentBasedRecommender =
field(default_factory=ContentBasedRecommender)
    collaborative: CollaborativeRecommender =
field(default_factory=CollaborativeRecommender)

    def fit_content(self, catalog: pd.DataFrame) -> None:
        self.content.fit(catalog)

    def fit_collab(self, interactions: pd.DataFrame, known_car_ids: Iterable[str]) ->
None:
        self.collaborative.fit(interactions, known_car_ids)

    def recommend(
        self,
        *,
        user_id: str | None,
        seed_car_ids: Iterable[str] | None = None,
        top_k: int = 5,
        exclude: Iterable[str] | None = None,
    ) -> list[RecommendedCar]:
        if not self.content.car_ids:
            return []

        seed = list(seed_car_ids or [])
        excluded = set(exclude or []) | set(seed)
```

```
        content_scores = dict(self.content.recommend(seed,
top_k=len(self.content.car_ids), exclude=excluded)) if seed else {}
        collab_scores: dict[str, float] = {}
        collab_reason = "popularity-fallback"

        if user_id and user_id in self.collaborative.user_idx:
            collab_scores = dict(
                self.collaborative.recommend(user_id, top_k=len(self.content.car_ids),
exclude=excluded)
            )
            collab_reason = "collaborative"
        elif self.collaborative.popularity:
            for cid, score in self.collaborative.top_popular(top_k=len(self.content.car_ids),
exclude=excluded):
                collab_scores[cid] = score

        candidates = set(content_scores) | set(collab_scores)
        if not candidates and seed:
            return [
                RecommendedCar(car_id=cid, score=score, reason="content-similarity")
                for cid, score in self.content.recommend(seed, top_k=top_k,
exclude=excluded)
            ]
        if not candidates:
            return [
                RecommendedCar(car_id=cid, score=score, reason="popularity")
                for cid, score in self.collaborative.top_popular(top_k=top_k,
exclude=excluded)
            ]

        alpha = self.alpha
        scored: list[tuple[str, float, str]] = []
        for cid in candidates:
            cs = content_scores.get(cid, 0.0)
            ks = collab_scores.get(cid, 0.0)
            score = alpha * cs + (1 - alpha) * ks
            reason = self._explain(cs, ks, collab_reason)
            scored.append((cid, score, reason))

        scored.sort(key=lambda item: -item[1])
        return [
            RecommendedCar(car_id=cid, score=float(max(0.0, min(1.0, score))),
reason=reason)
            for cid, score, reason in scored[:top_k]
```

```
    ]

@staticmethod
def _explain(content_score: float, collab_score: float, collab_reason: str) -> str:
    if content_score > 0 and collab_score > 0:
        return f"hybrid:content+{collab_reason}"
    if content_score > 0:
        return "content-similarity"
    return collab_reason
```

Додаток Б

Програмний код моделі LightGBM для прогнозування

```
from __future__ import annotations


from dataclasses import dataclass, field
from typing import Iterable

import lightgbm as lgb
import numpy as np
import pandas as pd

LAGS: tuple[int, ...] = (1, 2, 7, 14, 28)
ROLLING_WINDOWS: tuple[int, ...] = (7, 14)

def _calendar_features(idx: pd.DatetimeIndex) -> pd.DataFrame:
    return pd.DataFrame(
        {
            "dayofweek": idx.dayofweek,
            "month": idx.month,
            "dayofmonth": idx.day,
            "weekofyear": idx.isocalendar().week.astype(int).to_numpy(),
            "is_weekend": (idx.dayofweek >= 5).astype(int),
        },
        index=idx,
    )

def _lag_features(series: pd.Series, lags: Iterable[int] = LAGS) -> pd.DataFrame:
    frame = pd.DataFrame(index=series.index)
    for lag in lags:
        frame[f"lag_{lag}"] = series.shift(lag)
    for window in ROLLING_WINDOWS:
        shifted = series.shift(1)
        frame[f"roll_mean_{window}"] = shifted.rolling(window).mean()
        frame[f"roll_std_{window}"] = shifted.rolling(window).std()
    return frame

@dataclass
class LGBMForecaster:
    n_estimators: int = 400
    learning_rate: float = 0.05
    num_leaves: int = 31
```

```
    min_data_in_leaf: int = 5
    feature_fraction: float = 0.9
    bagging_fraction: float = 0.9

    _model: lgb.Booster | None = field(default=None, init=False, repr=False)
    _feature_columns: list[str] = field(default_factory=list, init=False, repr=False)
    _history: pd.Series = field(default_factory=lambda: pd.Series(dtype=float),
init=False, repr=False)

    def fit(self, series: pd.Series) -> "LGBMForecaster":
        history = series.astype(float).copy()
        history.index = pd.DatetimeIndex(history.index).normalize()
        self._history = history

        lag_frame = _lag_features(history)
        cal_frame = _calendar_features(history.index)
        frame = pd.concat([lag_frame, cal_frame], axis=1)
        frame["y"] = history.values

        frame = frame.dropna()
        if frame.empty:
            raise ValueError("Not enough history to compute LGBM features")

        self._feature_columns = [c for c in frame.columns if c != "y"]
        train_x = frame[self._feature_columns]
        train_y = frame["y"]

        dataset = lgb.Dataset(train_x, label=train_y, free_raw_data=False)
        params = {
            "objective": "regression_l1",
            "metric": "mae",
            "learning_rate": self.learning_rate,
            "num_leaves": self.num_leaves,
            "min_data_in_leaf": self.min_data_in_leaf,
            "feature_fraction": self.feature_fraction,
            "bagging_fraction": self.bagging_fraction,
            "bagging_freq": 5,
            "verbose": -1,
        }
        self._model = lgb.train(params, dataset, num_boost_round=self.n_estimators)
        return self

    def predict(self, horizon: int) -> pd.Series:
        if self._model is None:
```

```
            raise RuntimeError("LGBMForecaster.predict called before fit")

        history = self._history.copy()
        last_date = history.index[-1]
        future_index = pd.date_range(last_date + pd.Timedelta(days=1), periods=horizon,
freq="D")
        predictions: list[float] = []

        rolling = history.copy()
        for current_date in future_index:
            placeholder = pd.Series([np.nan], index=pd.DatetimeIndex([current_date]))
            rolling = pd.concat([rolling, placeholder])
            rolling.index = pd.DatetimeIndex(rolling.index)
            lag_frame = _lag_features(rolling)
            cal_frame = _calendar_features(rolling.index)
            features = pd.concat([lag_frame, cal_frame], axis=1).loc[[current_date]]
            features = features[self._feature_columns].ffill().fillna(0.0)
            yhat = float(self._model.predict(features)[0])
            yhat = max(0.0, yhat)
            predictions.append(yhat)
            rolling.loc[current_date] = yhat

        return pd.Series(predictions, index=future_index, name="count")
```

Додаток В

Копія опублікованих результатів