

2026 p.

Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____
Кафедра _____ Штучного інтелекту _____
Рівень вищої освіти _____ перший (бакалаврський) _____
Спеціальність _____ 122 Комп'ютерні науки _____
(код і повна назва)
Тип програми _____ освітньо-професійна _____
Освітня програма _____ Штучний інтелект _____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____» _____ 20 ____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві _____ Будзину Станіславу Ігоровичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка інтелектуальної автоматизованої системи для бронювання туристичного житла

затверджена наказом університету від 5 червня 2026 р. № 488Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 7 липня 2026 р.

3. Вихідні дані до роботи Науково-технічні публікації, дані Інтернет-джерел та відомих наукових і комерційних проєктів, .NET documentation, набір даних для тренування та тестування системи.

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Теоретичні основи побудови системи бронювання апартаментів _____

2) Аналіз вимог та проєктування системи бронювання апартаментів _____

3) Застосування штучного інтелекту в системі бронювання апартаментів _____

4) Програмна реалізація системи бронювання апартаментів _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.06.2026	виконано
2	Аналіз предметної галузі	09.06.2026	виконано
3	Огляд існуючих систем бронювання апартаментів	09.06.2026	виконано
4	Аналіз вимог до системи бронювання	10.06.2026	виконано
5	Проектування системи бронювання	11.06.2026	виконано
6	Туристичного житла		
7	Написання пояснювальної записки	16.06.2026	виконано
8	Перевірка на академічний плагіат	23.06.2026	виконано
9	Нормоконтроль	26.06.2026	виконано
10	Підготовка презентації та доповіді	31.06.2026	виконано
11	Попередній захист	02.07.2026	виконано
12	Рецензування	04.07.2026	виконано
13	Захист перед ЕК	07.07.2026	виконано

Дата видачі завдання 8 червня 20 26 р.

Здобувач _____
(підпис)

Керівник роботи _____
(підпис)

ас. Микола ЧЕРНЕНКО _____
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 91 с., 16 рис., 1 табл., 1 дод., 25 джерел.

ВЕКТОРНІ ПРЕДСТАВЛЕННЯ, ЗВАЖЕНЕ РАНЖУВАННЯ, КОРОТКОСТРОКОВА ОRENDA ЖИТЛА, ПЕРСОНАЛІЗАЦІЯ, СЕМАНТИЧНИЙ ПОШУК, СИСТЕМА БРОНЮВАННЯ АПАРТАМЕНТІВ, ШТУЧНИЙ ІНТЕЛЕКТ, ASP.NET CORE, HNSW, PGVECTOR, POSTGRESQL, REACT, RULE-BASED ФІЛЬТРАЦІЯ.

Об'єкт дослідження – задача відбору, ранжування та бронювання об'єктів нерухомості в інформаційних системах короткострокової оренди житла.

Предмет дослідження – методи rule-based фільтрації, багатокритеріального зваженого ранжування та семантичного пошуку на основі векторних представлень тексту для підвищення релевантності й персоналізації результатів пошуку.

Мета роботи – створення інформаційної системи бронювання апартаментів з механізмом інтелектуального відбору та ранжування об'єктів.

Методи дослідження – теоретичні (аналіз наукових джерел та систематизація існуючих підходів до фільтрації, ранжування й семантичного пошуку), експериментальний (програмна реалізація алгоритму ранжування, підсистеми семантичного пошуку та повноцінного веб-застосунку бронювання апартаментів). Методи розробки базуються на технологіях ASP.NET Core 8, PostgreSQL, React з TypeScript.

У результаті роботи проведено теоретичний аналіз підходів до побудови систем бронювання апартаментів, методів rule-based фільтрації та багатокритеріального зваженого ранжування, а також альтернативних методів персоналізації і метрик якості для їх оцінювання.

ABSTRACT

Bachelor's thesis contains: 91 pp., 16 fig., 1 tabl., 1 ann., 25 references.

APARTMENT BOOKING SYSTEM, ARTIFICIAL INTELLIGENCE, ASP.NET CORE, HNSW, PERSONALIZATION, PGVECTOR, POSTGRESQL, REACT, RULE-BASED FILTERING, SEMANTIC SEARCH, SHORT-TERM RENTAL, VECTOR EMBEDDINGS, WEIGHTED RANKING.

Object of research – the problem of selecting, ranking, and booking real estate properties in short-term rental information systems.

Subject of research – rule-based filtering methods, multi-criteria weighted ranking techniques, and semantic search methods based on vector text representations aimed at improving the relevance and personalization of search results.

The aim of the work – to develop an apartment booking information system with an intelligent property selection and ranking mechanism.

Research methods – theoretical methods (analysis of scientific literature and systematization of existing approaches to filtering, ranking, and semantic search), experimental (software implementation of the ranking algorithm, the semantic search subsystem, and a full-fledged apartment booking web application). The development methods are based on ASP.NET Core 8, PostgreSQL and React with TypeScript.

As a result of the research, a theoretical analysis of approaches to building apartment booking systems, methods of rule-based filtering and multi-criteria weighted ranking techniques was conducted. The alternative personalization methods and quality metrics for their evaluation were also analyzed.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	9
Вступ.....	11
1 Теоретичні основи побудови системи бронювання апартаментів.....	13
1.1 Ринок короткострокової оренди та проблематика цифрових платформ	13
1.2 Інформаційні системи бронювання: функціонал та еволюція	14
1.2.1 Еволюція інформаційних систем	14
1.2.2 Функціональне ядро системи	15
1.3 Алгоритмічні підходи до відбору та ранжування апартаментів	16
1.3.1 Логіка на основі правил	16
1.3.2 Ранжування та рекомендації як інструмент впорядкування результатів.....	18
1.3.3 Поєднання правил і ранжування у єдиному пайплайні пошуку	19
1.4 Ранжування та рекомендації як інструмент впорядкування результатів	20
1.5 Аналіз існуючих рішень у сфері бронювання апартаментів	21
1.5.1 Огляд комерційних платформ	21
1.5.2 Типові недоліки існуючих систем	22
1.6 Вимоги до системи бронювання апартаментів, що розробляється....	24
1.6.1 Функціональні вимоги	24
1.6.2 Нефункціональні вимоги	25
2 Аналіз вимог та проєктування системи бронювання апартаментів	27
2.1 Постановка задачі та мета роботи.....	27
2.2 Архітектура системи	28
2.2.1 Загальні принципи побудови	28
2.2.2 Клієнтський рівень	29
2.2.3 Серверний рівень та модульна структура.....	30
2.2.4 Рівень зберігання даних.....	31

2.3	Модель даних.....	32
2.3.1	Основні сутності та їх атрибути	32
2.3.2	Ключові зв'язки між сутностями.....	34
2.4	Алгоритм відбору та ранжування	35
2.4.1	Rule-based фільтрація: правила та послідовності перевірок	35
2.4.2	Алгоритм зваженого ранжування	37
2.4.3	Обробка конфліктних ситуацій при бронюванні	38
2.5	Технічний стек.....	39
2.5.1	Серверна частина.....	39
2.5.2	Клієнтська частина.....	42
2.5.3	Інфраструктура та розгортання.....	42
2.6	Заплановані результати.....	43
3	Застосування штучного інтелекту в системі бронювання апартаментів .	45
3.1	Місце штучного інтелекту в архітектурі системи	45
3.2	Семантичний пошук на основі векторних представлень тексту	46
3.2.1	Обмеження класичного пошуку за ключовими словами	46
3.2.2	Принцип роботи векторних представлень тексту	47
3.2.3	Технічна реалізація: pgvector та openai embeddings api.....	48
3.2.4	Інтеграція семантичного пошуку в пайплайн ранжування	49
3.3	Персоналізація ранжування на основі профілю користувача	50
3.3.1	Модель вподобань користувача	50
3.3.2	Адаптація вагових коефіцієнтів.....	51
3.3.3	Проблема холодного старту та її вирішення	52
3.4	Автоматична оцінка якості оголошень.....	53
3.4.1	Вплив якості опису на ефективність системи	53
3.4.2	Реалізація оцінювання засобами мовної моделі.....	53
3.4.3	Вплив оцінки якості на ранжування та поведінку орендодавців	54
3.5	Рекомендації схожих об'єктів	55
3.6	Оцінювання адекватності системи штучного інтелекту.....	56

3.7 Обмеження ai-компонентів та заходи їх мінімізації	58
4 Програмна реалізація системи бронювання апартаментів	61
4.1 Структура проєкту та архітектура рішення на рівні коду	61
4.2 Доменна модель та структура бази даних.....	63
4.3 Реалізації автентифікації та управління сесіями	65
4.4 Конвеєр пошуку: реалізація ланцюга фільтрів.....	68
4.5 Алгоритм ранжування: конкретна реалізація	70
4.6 Семантичний пошук: інтеграція openai та pgvector	72
4.7 Підсистема бронювання та обробка конфліктів.....	74
4.8 Фонові задачі та планувальник hangfire	77
4.9 Клієнтський застосунок	77
4.10 Контейнеризація та розгортання системи	81
4.11 Висновки по розділу.....	82
Висновки.....	84
Перелік джерел посилання	88
Додаток А Відомість кваліфікаційної роботи.....	91

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

ПЗ – програмне забезпечення;

СУБД – система управління базами даних;

AI – Artificial Intelligence – штучний інтелект;

ANN – Approximate Nearest Neighbor – наближений пошук найближчих сусідів;

API – Application Programming Interface – програмний інтерфейс застосунку;

BPMN – Business Process Model and Notation – нотація моделювання бізнес-процесів;

CQRS – Command Query Responsibility Segregation – розділення відповідальності команд і запитів;

CRUD – Create, Read, Update, Delete – створення, читання, оновлення, видалення;

DTO – Data Transfer Object – об'єкт передачі даних;

EF Core – Entity Framework Core – об'єктно-реляційний фреймворк для платформи .NET;

ER-діаграма – Entity-Relationship diagram – діаграма «сутність-зв'язок»;

GPT – Generative Pre-trained Transformer – архітектура генеративної попередньо навченої мовної моделі;

HNSW – Hierarchical Navigable Small World – ієрархічний навігаційний граф малих світів (алгоритм наближеного пошуку найближчих сусідів у векторному просторі);

HTTP/HTTPS – HyperText Transfer Protocol (Secure) – протокол передачі гіпертексту (захищений);

IDE – Integrated Development Environment – інтегроване середовище розробки;

JSON – JavaScript Object Notation – текстовий формат представлення структурованих даних;

JSONB – Binary JSON – бінарний формат зберігання JSON-даних у PostgreSQL;

JWT – JSON Web Token – токен автентифікації та авторизації у форматі JSON;

LINQ – Language Integrated Query – вбудована в мову C#/.NET технологія формування запитів до даних;

MVP – Minimal Viable Product – мінімально життєздатний продукт;

ORM – Object-Relational Mapping – об'єктно-реляційне відображення даних;

PMS – Property Management Systems – системи управління нерухомістю;

REST/RESTful API – Representational State Transfer – архітектурний стиль побудови програмних інтерфейсів;

SAW – Simple Additive Weighting – метод простої адитивної зваженості;

SDK – Software Development Kit – набір засобів для розробки програмного забезпечення;

SQL – Structured Query Language – структурована мова запитів до баз даних;

SRID – Spatial Reference Identifier – ідентифікатор системи просторових координат;

TLS – Transport Layer Security – протокол захисту транспортного рівня;

UI – User Interface – інтерфейс користувача;

WGS-84 – World Geodetic System 1984 – всесвітня геодезична система координат 1984 року.

ВСТУП

Цифровізація сфери послуг суттєво змінила ринок короткострокової оренди житла. Поява платформ, що дозволяють приватним власникам нерухомості пропонувати своє майно широкому колу потенційних гостей, перетворила цей сегмент з обмеженого переліку готелів та агентств на децентралізований ринок із мільйонами пропозицій. Подібне масштабування відкрило нові можливості для всіх учасників ринку, але водночас породило принципово нові виклики, пов'язані з надміром неструктурованої інформації, непрозорістю алгоритмів відбору та відсутністю ефективних інструментів для прийняття обґрунтованих рішень.

Користувач сучасної платформи бронювання зіштовхується з сотнями результатів пошуку, що відповідають базовим критеріям запиту, але не гарантують відповідності його індивідуальним потребам. Механізми ранжування існуючих платформ, як правило, не є прозорими для користувача, а порядок видачі результатів може визначатися комерційними інтересами платформи замість реальної відповідності запиту. Відсутність персоналізації змушує орендаря щоразу заново встановлювати параметри пошуку без урахування попереднього досвіду взаємодії зі сервісом.

У науковій та практичній площині вирішення зазначених проблем пов'язане з розробкою інтелектуальних систем відбору та ранжування, що здатні обробляти різномірні дані про об'єкти нерухомості, застосовувати детерміністичні правила фільтрації та впорядковувати результати за сукупністю зважених критеріїв. Такий підхід відповідає загальній тенденції до автоматизації аналітичних задач у сервісній індустрії і є активним напрямом досліджень у сфері інформаційних систем та систем підтримки прийняття рішень.

Метою кваліфікаційної роботи є дослідження предметної галузі короткострокової оренди житла, аналіз проблем наявних платформ бронювання та обґрунтування концепції системи бронювання апартаментів

з механізмами інтелектуального відбору та ранжування пропозицій. У рамках практики проводиться огляд та порівняльний аналіз існуючих рішень, виявляються їхні ключові недоліки та формулюється постановка задачі для подальшої реалізації інформаційної системи.

Практична значущість роботи полягає у тому, що розроблювана система може бути використана як самостійна платформа для бронювання апартаментів, де всі рішення щодо архітектури та алгоритмів ранжування є детально обґрунтованими і можуть бути перевірені. Така прозорість є принциповою відмінністю від комерційних платформ, де алгоритми ранжування є закритими і можуть підпорядковуватися комерційним інтересам.

1 ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ СИСТЕМИ БРОНІЮВАННЯ АПАРТАМЕНТІВ

1.1 Ринок короткострокової оренди та проблематика цифрових платформ

Короткострокова оренда житла перетворилася на окремий сегмент туристичної та готельної індустрії завдяки появі платформ, що дозволяють приватним орендодавцям пропонувати своє майно широкому колу потенційних гостей. Йдеться не просто про технологічний зсув у способі пошуку житла, змінилася сама природа ринку. Замість обмеженої кількості готелів та агентств нерухомості з'явився децентралізований простір пропозицій, де кожен об'єкт має власні умови, правила та характеристики. Пірингові ринки, відомі під загальною назвою економіка спільного споживання, виникли як альтернативні постачальники товарів і послуг, що традиційно надавалися давно усталеними галузями [1]. Технологічні інновації спростили процес виходу на ринок для постачальників, забезпечили зручний пошук оголошень для споживачів та знизили транзакційні витрати [1]. Такий перехід відкрив нові можливості для мандрівників, але водночас суттєво ускладнив завдання вибору.

Коли кількість пропозицій на одному майданчику вимірюється тисячами, а на великих платформах мільйонами стандартні підходи до пошуку та фільтрації стають недостатніми. Користувач може задати місто, дати та кількість гостей і отримати сотні результатів, серед яких доведеться самотійно розбиратися. Проблема полягає не у відсутності варіантів, а в їх надлишку без належної структури. Платформа відображає все, що відповідає запиту, але не допомагає зрозуміти, що з цього дійсно варте уваги і чому.

Окремою складністю є неоднорідність даних, що описують об'єкти нерухомості. Різні орендодавці по-різному заповнюють картки

апартаментів. Одні вказують детальний опис кожної кімнати, а інші обмежуються кількома рядками. Набір зручностей, правила заселення, умови скасування та фотографії залежать від конкретного господаря, а не від стандарту платформи. Такий різномірний масив даних важко порівнювати. Методи на основі контенту обмежені набором ознак, явно закріплених за об'єктами, а два різні об'єкти, представлені однаковим набором ознак, стають нерозрізнюваними [2]. Саме тут починаються реальні труднощі для будь-якої системи, що прагне запропонувати корисне ранжування результатів.

Динамічне ціноутворення додає ще один шар складності. Вартість одних і тих самих апартаментів у різний час може відрізнятися в рази. Це робить ціну змінною величиною. Система бронювання, яка не враховує цей аспект, ризикує видавати результати, що вводять користувача в оману щодо реальної вартості проживання.

Варто зазначити, що більшість наявних рішень орієнтовані на середньостатистичного користувача без урахування його індивідуального контексту. Ділова поїздка на два дні чи сімейний чи груповий відпочинок це принципово різні сценарії з різними пріоритетами щодо розташування, розміру, зручностей та ціни. Просто рекомендувати об'єкти користувачам може бути недостатньо, адже рекомендаційна система має враховувати додаткову контекстну інформацію, таку як час, місце та оточення користувача [2]. Платформа, яка не розрізняє ці випадки, змушує самого орендаря щоразу налаштовувати пошук із нуля.

1.2 Інформаційні системи бронювання: функціонал та еволюція

1.2.1 Еволюція інформаційних систем

Перші цифрові системи бронювання житла фактично відтворювали логіку паперового каталогу. Користувач отримував перелік доступних

об'єктів і самостійно приймав рішення. Такий підхід цілком виправдовував себе, коли пропозицій було мало, а вибір зводився до кількох очевидних критеріїв. Із масштабуванням ринку та зростанням кількості об'єктів пасивна модель відображення даних перестала відповідати потребам користувачів. Замість спрощення вибору платформа перетворилася на додаткове джерело плутанини.

Сучасна інформаційна система бронювання апартаментів це більше ніж просто база даних із пошуком. Вона виступає активним посередником між запитом користувача та масивом доступних пропозицій, беручи на себе частину аналітичної роботи. Замість того щоб просто передати клієнту необроблені результати, система оцінює варіанти, встановлює між ними ієрархію та пропонує ті, що найбільше відповідають виявленому контексту. Power характеризує цю еволюцію як перехід від пасивних інформаційних систем до активних систем підтримки прийняття рішень, де завдання системи полягає у зменшенні інформаційного навантаження та структуруванні вибору. Саме цей перехід є ключовою відмінністю між системами першого та поточного покоління.

У науковій літературі подібні рішення розглядаються як різновид систем підтримки прийняття рішень (Decision Support Systems), де завдання системи полягає не у підміні людського судження, а у зменшенні інформаційного навантаження та структуруванні вибору. Стосовно бронювання житла це означає, що система не вирішує за користувача, де зупинитися, але суттєво звужує простір альтернатив до тих, що мають реальний сенс у конкретній ситуації.

1.2.2 Функціональне ядро системи

Функціональність системи бронювання апартаментів визначається не лише набором технічних можливостей, а й логікою їх взаємодії між собою. Розрізнені функції, а саме пошук, фільтрація, ранжування, бронювання,

оплата набувають цінності лише тоді, коли організовані у зв'язний процес, де кожен крок готує ґрунт для наступного. З цієї точки зору ключовими складовими є:

- збір та нормалізація даних про об'єкти нерухомості з різних джерел у єдиний внутрішній формат;
- перевірка доступності об'єктів на задані дати з урахуванням поточних бронювань і блокувань календаря;
- відсіювання пропозицій, що не відповідають базовим параметрам запиту, ще до етапу ранжування;
- впорядкування відібраних варіантів за сукупністю критеріїв з можливістю адаптації до профілю користувача;
- фіксація вибору клієнта та підтвердження бронювання з перевіркою актуальності даних на момент оформлення;
- обробка платіжної транзакції та збереження деталей замовлення для обох сторін угоди.

Важливо розуміти, що перелічені функції не є рівнозначними з точки зору складності реалізації та впливу на якість сервісу. Фіксація бронювання та обробка платежів це операційні задачі з чіткою логікою і відносно невеликим простором для варіацій. Натомість відсіювання та ранжування є інтелектуальними задачами, де якість результату безпосередньо визначає корисність платформи для кінцевого користувача. Саме тому ці два компоненти заслуговують на окрему увагу при проєктуванні системи.

1.3 Алгоритмічні підходи до відбору та ранжування апартаментів

1.3.1 Логіка на основі правил

У темі пошуку апартаментів правила є найбільш природним способом визначити обмеження. Якщо орендар вказав, що потребує апартаменти на п'ять осіб, об'єкт на чотири місця просто не може бути релевантним,

незалежно від його рейтингу чи ціни. Якщо задані дати заїзду і виїзду, то апартаменти, заброньовані на цей період, мають бути просто виключені з будь-яких подальших дій. Саме такого роду беззаперечні умови і описуються через правила.

Rule-based фільтрація виконується першою у ланцюжку обробки запиту і виступає як попередня вибірка, що різко скорочує кількість кандидатів для подальшого аналізу. Типові правила у системі бронювання апартаментів охоплюють такі перевірки:

- відповідність доступності об'єкта у вказаному діапазоні дат без перетину з наявними бронюваннями;
- відповідність максимальної місткості апартаментів кількості гостей, зазначеній у запиті;
- перебування ціни за добу в межах заданого цінового діапазону;
- наявність обов'язкових зручностей, відмічених користувачем як обов'язкові умови;
- мінімальний строк оренди, якщо орендодавець встановив таке обмеження;
- географічна відповідність (перебування об'єкта у заданому районі або в межах встановленого радіуса);
- відповідність середнього рейтингу об'єкта мінімальному значенню, заданому користувачем у критеріях пошуку.

Принципова перевага такого підходу полягає у тому, що його поведінка повністю передбачувана і пояснювана. Якщо апартаменти не потрапили до результатів, то завжди можна точно вказати, яке саме правило це зумовило. Це усуває будь-яку неоднозначність у роботі фільтра. Ця властивість є важливою як для налагодження системи розробниками, так і для довіри з боку користувачів: прозорість фільтрації збільшує розуміння системи і дозволяє орендарю розуміти, чому певні варіанти не відображаються [3].

1.3.2 Ранжування та рекомендації як інструмент впорядкування результатів

Після того як правила виключили нерелевантні об'єкти, перед системою постає задача іншого роду: серед тих, що залишилися, необхідно встановити порядок, що відображав би ступінь відповідності кожного варіанта потребам конкретного користувача. Це завдання принципово відрізняється від фільтрації, бо тут немає однозначних критеріїв, а є лише відносні переваги.

Зважене ранжування передбачає присвоєння кожному об'єкту числової оцінки на основі набору характеристик, кожна з яких має свій коефіцієнт важливості. Для системи бронювання апартаментів такими характеристиками можуть бути відносна вартість у порівнянні з аналогічними об'єктами в тому ж районі, середній рейтинг відгуків та їх кількість, відстань до ключових точок інтересу, рівень відповідності набору зручностей побажанням користувача. Підсумкова оцінка є інтегральним показником, що дозволяє порівнювати між собою апартаменти, які мають різні профілі переваг і недоліків. Цей підхід формалізується як метод простої адитивної зваженості, де оцінка альтернативи обчислюється як зважена сума нормалізованих значень критеріїв [4].

Персоналізація ранжування додає до цієї схеми ще один вимір. Коефіцієнти важливості критеріїв можуть змінюватися залежно від того, що відомо про конкретного користувача. Наприклад, якщо система бачить, що у попередніх бронюваннях він обирав об'єкти в центрі міста і ніколи не звертав уваги на різницю в ціні, то логічно підвищити вагу критерію розташування та знизити вагу ціни для цього профілю. Врахування індивідуальних вподобань при ранжуванні суттєво підвищує точність рекомендацій порівняно з неперсоналізованими моделями [5]. Навіть проста персоналізація на рівні явно збережених вподобань значно покращує якість результатів.

Окремо слід зазначити задачу відображення об'єктів у ситуаціях, де критерії перебувають у конфлікті між собою. Найдешевші апартаменти рідко мають найкраще розташування або найвище оціненими. Система не може оптимізувати одночасно за всіма параметрами, тому результатом ранжування є не ідеальний варіант, а впорядкована множина кандидатів, кожен з яких є оптимальним у певному сенсі. Прозоре відображення того, чому той чи інший об'єкт опинився вище або нижче у списку, допомагає користувачеві усвідомити природу цього компромісу. Крім того, прозорість алгоритму ранжування підвищує довіру користувача до системи та сприяє прийняттю обґрунтованих рішень [3].

1.3.3 Поєднання правил і ранжування у єдиному пайплайні пошуку

На практиці жоден з описаних підходів не є самодостатнім. Чиста система правил відсіює непридатні варіанти, але не здатна вибрати найкращий серед придатних, адже усі вони є рівнозначними з точки зору правил. Чисте ранжування без попередньої фільтрації ризикує включити до результатів об'єкти, що формально не відповідають умовам запиту, якщо алгоритм не навчений жорстко їх виключати. Поєднання двох підходів у послідовному пайплайні усуває слабкі місця кожного з них.

Логіка роботи гібридного пайплайну є каскадною. Спочатку правила виконують детерміністичне відсіювання, формуючи підмножину допустимих кандидатів, а потім алгоритм ранжування впорядковує їх за ступенем відповідності. Такий поділ чітко розмежовує зони відповідальності: правила гарантують коректність результатів, ранжування гарантує їх релевантність. Ряд систем використовують гібридний підхід, поєднуючи різні методи, що допомагає уникнути обмежень кожного з них окремо [2]. Такий поділ спрощує підтримку та розвиток системи з часом.

Додатковою перевагою гібридного підходу є те, що він дозволяє системі пояснювати свої рішення на різних рівнях деталізації. Для

виключення об'єкта завжди є конкретне правило. Для місця у ранжуванні є набір зважених критеріїв з відомими коефіцієнтами. Така пояснюваність є важливою не лише для зручності користувача, а й для можливості аудиту та налаштування системи розробниками і адміністраторами платформи.

1.4 Ранжування та рекомендації як інструмент впорядкування результатів

Цінність системи бронювання апартаментів не є однаковою для всіх, хто з нею взаємодіє. Орендар, орендодавець і сама платформа отримують різні вигоди, а тому розуміння цих відмінностей важливе для того, щоб правильно розставити пріоритети при проєктуванні.

Для орендаря ключовим є скорочення часу і зусиль, необхідних для прийняття рішення. Коли система коректно відсіює нерелевантні варіанти і виводить на першу позицію те, що дійсно підходить. Користувач може знайти відповідне житло за кілька хвилин замість декількох годин самостійного порівняння. Ця різниця не є абстрактною зручністю: у сегменті короткострокової оренди ціни часто змінюються, і швидкість прийняття рішення безпосередньо впливає на підсумкову вартість бронювання. Крім того, цінова реакція на ринку короткострокової оренди є особливо вираженою в періоди пікового попиту [1], що підтверджує значення швидкості прийняття рішення для підсумкової вартості бронювання.

Для орендодавця інтелектуальна система означає вищу видимість серед цільової аудиторії. Алгоритмічне ранжування, на відміну від простого хронологічного або алфавітного сортування, виводить об'єкти до тих користувачів, для яких вони є дійсно релевантними. На прикладі Amazon було продемонстровано, що алгоритмічне ранжування, яке показує релевантні пропозиції, суттєво підвищує конверсію переглядів у транзакції [6].

Для платформи як бізнесу інтелектуальна система є бізнес інструментом, що допомагає отримати свою частку ринку та автоматизувати процеси. Замість того, щоб покладатися на ручне налаштування видачі або прості фіксовані алгоритми, платформа отримує механізм, що адаптується до нових даних і сценаріїв без необхідності переписувати логіку з нуля. Це особливо важливо при масштабуванні: система, що добре працює з тисячею об'єктів, має залишатися ефективною і при мільйоні без пропорційного зростання витрат на підтримку.

Варто також відзначити ширший контекст, адже розвиток інтелектуальних систем бронювання відповідає загальній тенденції до автоматизації рутинних аналітичних задач у сервісній індустрії. Те, що раніше вимагало участі людини-консультанта або тривалого самостійного дослідження, сьогодні може бути делеговане алгоритму, але обов'язково за умови, що цей алгоритм достатньо точно моделює логіку обґрунтованого вибору. Розробка такої системи є не лише технічною, але й дослідницькою задачею, що потребує ретельного аналізу предметної галузі та потреб усіх учасників процесу оренди.

1.5 Аналіз існуючих рішень у сфері бронювання апартаментів

1.5.1 Огляд комерційних платформ

Сучасний ринок цифрових сервісів бронювання апартаментів представлений кількома категоріями рішень, що відрізняються за моделлю роботи, функціональністю та цільовою аудиторією. Аналіз наявних платформ дозволяє виявити типові підходи до організації пошуку та відбору об'єктів, а також окреслити ті аспекти, у яких існуючі рішення залишають простір для вдосконалення.

Найпоширенішим типом є великі агрегаторні платформи такі, як Airbnb, Booking.com, Vrbo та аналогічні сервіси. Їхня сильна сторона

масштаб: мільйони об'єктів, глобальне охоплення, розвинена інфраструктура платежів і комунікації між орендарем та власником. Логіка пошуку на таких платформах найчастіше поєднує базову фільтрацію за параметрами з алгоритмічним ранжуванням, що враховує ймовірність бронювання, активність власника та історію відгуків. Водночас конкретні алгоритми не розкриваються, що ускладнює аналіз їхньої роботи з наукової точки зору. В дослідженнях Airbnb характеризується як деструктивна інновація, що сформувала неформальний сегмент туристичного розміщення [3], проте конкретні алгоритми ранжування таких платформ залишаються непрозорими, що ускладнює їх незалежний аналіз [7].

Окрему категорію становлять спеціалізовані платформи та регіональні агрегатори, що зосереджуються на конкретному типі нерухомості або географії. Такі сервіси, як правило, пропонують глибшу деталізацію об'єктів і більш точні фільтри, але поступаються великим платформам за кількістю пропозицій і впізнаваністю бренду. Їхня перевага у кращому розумінні специфіки локального ринку та часто у більш прозорій логіці ранжування.

Третю категорію складають системи управління нерухомістю (Property Management Systems, PMS), орієнтовані на сторону орендодавця. Ці рішення вирішують задачу управління календарем доступності, синхронізації з кількома платформами одночасно та автоматизації комунікації з гостями. Для орендаря вони здебільшого невидимі, але безпосередньо впливають на якість даних, що надходять до агрегаторів [8].

1.5.2 Типові недоліки існуючих систем

Незважаючи на розвиненість великих платформ, аналіз їх роботи з позиції користувача виявляє ряд системних недоліків, що залишаються актуальними незалежно від масштабу сервісу.

Перша проблема – непрозорість алгоритму ранжування. Користувач бачить результат, але не розуміє, чому саме ці об'єкти опинилися на перших позиціях. На практиці це означає, що позиція в результатах пошуку може залежати не лише від відповідності запиту, але й від комерційних відносин між платформою та власником, статусу або підключення до платних програм просування. Відсутність прозорості ускладнює довіру до результатів і не дозволяє орендарю зрозуміти, які саме його вимоги були пріоритизовані [3].

Друга проблема це обмеженість персоналізації. Більшість платформ пропонують однаковий порядок результатів для всіх користувачів, що здійснюють ідентичний пошуковий запит, незалежно від їхньої попередньої поведінки або вираженого профілю вподобань. Виняток складають окремі функції, як збереження об'єктів до вибраного або налаштування цінових сповіщень, однак ці дані рідко систематично впливають на порядок видачі результатів нового пошуку.

Третя проблема це фрагментованість інформації. Навіть у межах однієї платформи дані про об'єкти нерідко є неповними або структурованими по-різному. Один власник детально описує кожен зручність, інший залишає поля пустими або вказує інформацію в довільному текстовому описі, що алгоритм не може автоматично проаналізувати. Це призводить до ситуацій, коли об'єкти з недостатньо заповненим профілем потрапляють у результати пошуку поряд із повністю описаними і орендар змушений самотійно доповнювати картину через читання відгуків або уточнюючі питання власнику [2].

Четверта проблема – слабка підтримка складних сценаріїв бронювання. Стандартні фільтри більшості платформ добре справляються з простими запитам, однак коли запит ускладнюється існуючі фільтри або не покривають усі умови, або вимагають ручного перебору результатів. Автоматизована обробка таких складних умов залишається слабким місцем переважної більшості наявних рішень [2].

1.6 Вимоги до системи бронювання апартаментів, що розробляється

1.6.1 Функціональні вимоги

Аналіз предметної галузі та огляд існуючих рішень дозволяють сформулювати конкретні функціональні вимоги до системи, що розробляється. Ці вимоги відображають як базові очікування від будь-якого сервісу бронювання, так і специфічні завдання, вирішення яких відрізнятиме розроблювану систему від типових аналогів.

У частині управління об'єктами система має забезпечити повноцінний CRUD-функціонал для оголошень: створення, редагування, деактивацію та читання. Кожен об'єкт описується структурованим набором атрибутів: адреса та координати, тип нерухомості, місткість, перелік зручностей, фотографії, умови проживання, правила скасування та цінова модель. Окремо ведеться календар доступності, що дозволяє власнику блокувати дати незалежно від наявності активних бронювань.

У частині пошуку та відбору система має підтримувати пошуковий запит із параметрами:

- місто або географічна зона пошуку з підтримкою пошуку за радіусом від заданої точки;
- дати заселення та виселення з автоматичною перевіркою доступності;
- кількість гостей з урахуванням максимальної місткості об'єкта;
- ціновий діапазон із урахуванням усіх обов'язкових зборів;
- тип об'єкта та мінімальний набір обов'язкових зручностей;
- додаткові умови (мінімальний рейтинг, тип скасування, дозвіл на проживання з тваринами).

Результати пошуку мають відображатися у впорядкованому вигляді з можливістю перемикання між режимами сортування, а саме за релевантністю, за ціною, за рейтингом. Кожен об'єкт у результатах

представляється карткою з ключовими характеристиками, що дозволяє прийняти первинне рішення про доцільність детального перегляду без переходу на сторінку об'єкта.

У частині бронювання система забезпечує повний цикл: від вибору об'єкта до підтвердження резервування. Це включає перевірку актуальності доступності безпосередньо перед підтвердженням, збереження параметрів бронювання, генерацію підтвердження для обох сторін та оновлення календаря доступності. Система також підтримує скасування бронювань із автоматичним розрахунком повернення коштів відповідно до умов конкретного оголошення.

1.6.2 Нефункціональні вимоги

Окрім безпосереднього функціоналу, до системи висувуються вимоги, що стосуються якісних характеристик її роботи. Ці вимоги є не менш важливими. Система, що коректно виконує всі функції, але повільно відповідає або є незручною у використанні, не вирішить задачу, для якої призначена.

З точки зору продуктивності пошуковий запит із типовими параметрами має повертати результати не довше ніж за дві секунди за умов типового навантаження. Це обмеження зумовлене поведінкою користувачів: дослідження у сфері веб-юзабіліті показують, що затримка понад дві секунди помітно знижує залученість і підвищує відсоток відмов. Як зазначається в дослідженні, 1 секунда є межею збереження безперервного потоку думок користувача, а затримка понад 10 секунд – межею утримання уваги [10]. На практиці для пошукових систем прийнятним порогом вважається 2 секунди. Для забезпечення цього показника необхідне продумане індексування бази даних і ефективна організація запитів до таблиць із великою кількістю записів.

З точки зору надійності система має коректно обробляти помилкові вхідні дані, недоступність зовнішніх сервісів і одночасні запити на бронювання одного об'єкта. Останнє є критичним, бо ситуація подвійного бронювання, коли двоє користувачів одночасно підтверджують резервування того самого об'єкта на ті самі дати, неприпустима.

З точки зору безпеки система має забезпечувати аутентифікацію та авторизацію користувачів, розмежування прав доступу між ролями (орендар, орендодавець, адміністратор), захист персональних даних і безпечну передачу інформації. Зберігання паролів виключно у вигляді хешів є мінімальними базовими вимогами.

З точки зору розширюваності архітектура системи має допускати додавання нових функцій, тобто нових типів фільтрів, нових критеріїв ранжування, нових зовнішніх інтеграцій без необхідності переписувати існуючу логіку. Це досягається через чіткий поділ відповідальності між компонентами і мінімізацію залежностей між ними.

Було виконано аналіз предметної галузі короткострокової оренди житла та встановлено ключові проблеми, що обмежують ефективність наявних платформ, проведено огляд функціонального ядра сучасних систем бронювання та їхньої еволюції. Визначено, що задовільного рішення проблеми ранжування при великій кількості об'єктів без урахування семантики запиту та індивідуального контексту користувача бракує у більшості розглянутих систем. Сукупність функціональних і нефункціональних вимог утворює основу для проєктних рішень наступного розділу. Кожна архітектурна або технологічна вимога, що обґрунтовується далі, спирається на конкретні потреби, виявлені в ході аналізу предметної галузі та огляду існуючих рішень.

2 АНАЛІЗ ВИМОГ ТА ПРОЄКТУВАННЯ СИСТЕМИ БРОНЮВАННЯ АПАРТАМЕНТІВ

2.1 Постановка задачі та мета роботи

В першому розділі визначено ключові проблеми ринку короткострокової оренди: надлишок нефільтрованих пропозицій, неоднорідність даних, обмеженість персоналізації та слабка підтримка складних пошукових сценаріїв. Системи, що наразі існують на ринку, або не розкривають логіку ранжування, або підпорядковують її комерційним інтересам платформи. Крім того, серед широкого асортименту пропозицій, користувачу може часто бути важко знайти саме те, що було б релевантно для його запиту. Це формує технічне завдання: спроектувати систему бронювання апартаментів, де логіка відбору та впорядкування результатів є прозорою, обґрунтованою та відповідає реальним потребам орендаря, а пошук може бути здійсненим, опираючись на конкретні побажання користувача.

Метою кваліфікаційної роботи є розробка та реалізація інформаційної системи бронювання апартаментів із вбудованим механізмом відбору та ранжування об'єктів на основі поєднання rule-based фільтрації та зваженого ранжування, а також системою семантичного пошуку. Система має виступати посередником між масивом доступних пропозицій та конкретним запитом користувача, перекладаючи на себе аналітичну роботу, яку орендар зазвичай виконує вручну.

Для досягнення цієї мети необхідно вирішити такі задачі. По-перше, спроектувати архітектуру системи з чітким розподілом відповідальності між компонентами. По-друге, реалізувати модуль rule-based фільтрації, що відсіює об'єкти за детерміністичними критеріями до етапу ранжування. По-третє, побудувати алгоритм зваженого ранжування, що враховує кілька критеріїв і допускає персоналізацію. По-четверте, реалізувати можливості

для семантичного пошуку за описом. По-п'яте, забезпечити повний цикл бронювання, включно з перевіркою доступності, фіксацією резервування та обробкою скасувань. По-шосте, реалізувати управління об'єктами з боку орендодавця та адміністративний інтерфейс.

Система, що розробляється позиціонується як самостійна платформа, а не надбудова над існуючим агрегатором. Це означає, що всі рішення щодо архітектури, моделі даних та алгоритмів ранжування приймаються незалежно і можуть бути детально обґрунтовані. Така позиція дає можливість дослідити не лише технічну реалізацію, але й архітектурні компроміси, що виникають під час проектування систем подібного класу.

2.2 Архітектура системи

2.2.1 Загальні принципи побудови

Архітектура системи бронювання апартаментів визначає склад функціональних компонентів, їх взаємодію між собою, де зосереджена бізнес-логіка і яким чином система може розвиватися без деградації існуючих частин. Оскільки перший розділ сформулював конкретні вимоги (продуктивність пошукового запиту, надійність при одночасних бронюваннях, розширюваність фільтрів і критеріїв ранжування) архітектурні рішення мають бути прямою відповіддю на ці вимоги, а не абстрактним наслідуванням стандартних схем.

Для системи обрано трирівневу архітектуру з чітким поділом на клієнтський рівень, серверний рівень бізнес-логіки та рівень зберігання даних. Кожен рівень взаємодіє лише з суміжними і не має прямого доступу через рівень. Серверний рівень додатково поділяється на кілька функціональних компонентів, що відповідають за окремі аспекти обробки запиту. Такий поділ забезпечує можливість змінювати логіку ранжування

або правила фільтрації без впливу на клієнтський інтерфейс або схему бази даних.

Принциповим рішенням є розміщення всієї логіки відбору та ранжування на серверному рівні. Клієнтська частина проектується за принципом тонкого клієнта, не приймає жодних рішень щодо того, які об'єкти показувати і в якому порядку, лише передає параметри запиту і відображає готову відповідь. Це спрощує розробку клієнтської частини, дозволяє незалежно тестувати алгоритми на серверному рівні та гарантує однаковість логіки незалежно від того, через який інтерфейс надходить запит (веб браузер, мобільний застосунок або зовнішній API). На рисунку 2.1 зображена C4 model застосунку.

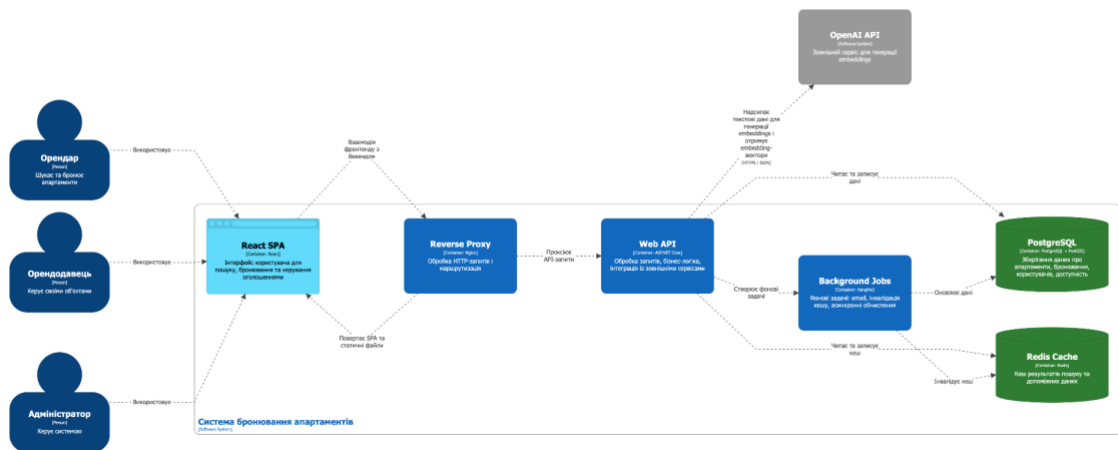


Рисунок 2.1 – C4 model застосунку

2.2.2 Клієнтський рівень

Клієнтський рівень реалізує інтерфейс для двох основних груп користувачів: орендарів і орендодавців. Для орендаря це форма пошуку з параметрами, список результатів, сторінка об'єкта та форма бронювання. Для орендодавця панель управління оголошеннями, редактор об'єкта та перегляд поточних бронювань.

Клієнтська частина не виконує аналітичних операцій. Будь-яка логіка, що стосується відбору або впорядкування результатів, знаходиться на серверному рівні. Клієнт надсилає структурований запит і відображає отриману відповідь. Єдиний виняток це клієнтська валідація введених даних: перевірка коректності дат, відповідність числових полів допустимим діапазнам тощо. Ця валідація виконується на клієнтській стороні виключно для покращення зручності використання і продубльована на серверному рівні як обов'язкова перевірка.

З точки зору відображення результатів ключовим є те, як клієнтська частина представляє ранжовані дані. Кожна картка об'єкта має демонструвати не лише базові характеристики, а й чому цей варіант опинився на поточній позиції: яка підсумкова оцінка, які критерії внесли найбільший внесок. Така прозорість є прямою відповіддю на проблему непрозорості ранжування, виявлену в аналізі існуючих платформ.

2.2.3 Серверний рівень та модульна структура

Серверний рівень є ядром системи і несе відповідальність за обробку всіх запитів, виконання бізнес-логіки та взаємодію з базою даних. Він організований як набір функціональних модулів із чітко визначеними інтерфейсами між ними. Кожен модуль вирішує одну задачу і не знає про внутрішню реалізацію суміжних.

Модуль обробки запитів є точкою входу для всіх звернень від клієнтської частини. Він відповідає за маршрутизацію, серверну валідацію вхідних даних та формування стандартизованої внутрішньої структури запиту. Якщо вхідні дані некоректні, запит відхиляється на цьому етапі з відповідним повідомленням про помилку.

Модуль rule-based фільтрації отримує стандартизований запит і виконує перший прохід по базі даних. Його задача сформувати підмножину об'єктів, що задовольняють усі обов'язкові умови запиту. Цей модуль не

виставляє оцінок і не порівнює об'єкти між собою. Він лише визначає, чи є конкретний об'єкт допустимим кандидатом. Результатом роботи модуля є набір ідентифікаторів об'єктів, переданих на наступний етап.

Модуль ранжування приймає набір допустимих кандидатів і впорядковує їх за підсумковою оцінкою. Оцінка розраховується як зважена сума нормалізованих значень кількох критеріїв. Коефіцієнти ваги можуть коригуватися залежно від профілю користувача, якщо така інформація є доступною. Модуль повертає впорядкований список із розрахованими оцінками та розбивкою внеску кожного критерію. Ці дані використовуються клієнтською частиною для відображення пояснення позиції об'єкта.

Модуль управління бронюваннями відповідає за весь операційний цикл резервування: перевірку актуальності доступності безпосередньо перед підтвердженням, атомарну фіксацію бронювання в базі даних, оновлення календаря доступності та обробку скасування із розрахунком повернення коштів. Цей модуль є критичним з точки зору транзакційної цілісності і проєктується з урахуванням можливості одночасних запитів на один об'єкт.

Модуль управління об'єктами забезпечує CRUD-операції для оголошень орендодавців: створення, редагування, зміну статусу доступності та керування календарем. Він також відповідає за нормалізацію даних при збереженні нового або оновленого об'єкта.

2.2.4 Рівень зберігання даних

Рівень зберігання даних будується на PostgreSQL. Вибір реляційної СУБД обґрунтований структурою даних предметної галузі: об'єкти нерухомості, їх характеристики, записи бронювань та профілі користувачів пов'язані між собою відношеннями, що природно описуються реляційною моделлю. Крім того, задача перевірки перетину дат бронювань ефективно

вирішується засобами SQL, тоді як в документоорієнтованих або ключ-значення сховищах це потребувало б складнішої логіки на рівні застосунку.

Схема бази даних організована навколо чотирьох основних сутностей. Перша – об'єкт нерухомості з усіма його атрибутами: адресою, координатами, типом, місткістю, набором зручностей, ціновою моделлю та умовами проживання. Друга – запис бронювання, що фіксує орендаря, об'єкт, дати, статус та деталі оплати. Третя – запис доступності, що дозволяє власнику блокувати конкретні дати незалежно від бронювань. Четверта – профіль користувача, де зберігаються дані аутентифікації, роль у системі та, за наявності, вподобання для персоналізації ранжування.

Продуктивність пошукового запиту, що є нефункціональною вимогою з першого розділу, забезпечується насамперед індексуванням. Пошукові запити найчастіше фільтрують за географічними координатами, діапазоном дат та місткістю, тому ці поля індексуються в першу чергу. Для перевірки перетину дат використовується SQL-конструкція, що дозволяє виявити конфліктні бронювання за один запит без ітеративної перевірки кожної дати окремо.

Для зменшення навантаження на базу даних при повторюваних пошукових запитах передбачено рівень кешування на основі Redis. Результати пошуку з однаковими або близькими параметрами зберігаються в кеші з обмеженим терміном дії. Запис нового бронювання або зміна доступності об'єкта автоматично інвалідує відповідні записи кешу, гарантуючи актуальність відображуваних даних.

2.3 Модель даних

2.3.1 Основні сутності та їх атрибути

Модель даних є фундаментом, на якому будується вся логіка системи. Її коректне проектування визначає, наскільки ефективно працюватимуть

фільтрація та ранжування і наскільки легко систему буде розширювати в майбутньому. Нижче описані основні сутності, їх ключові атрибути та обґрунтування прийнятих рішень щодо структури. Крім того, на рисунку 2.2 зображена ER діаграма.

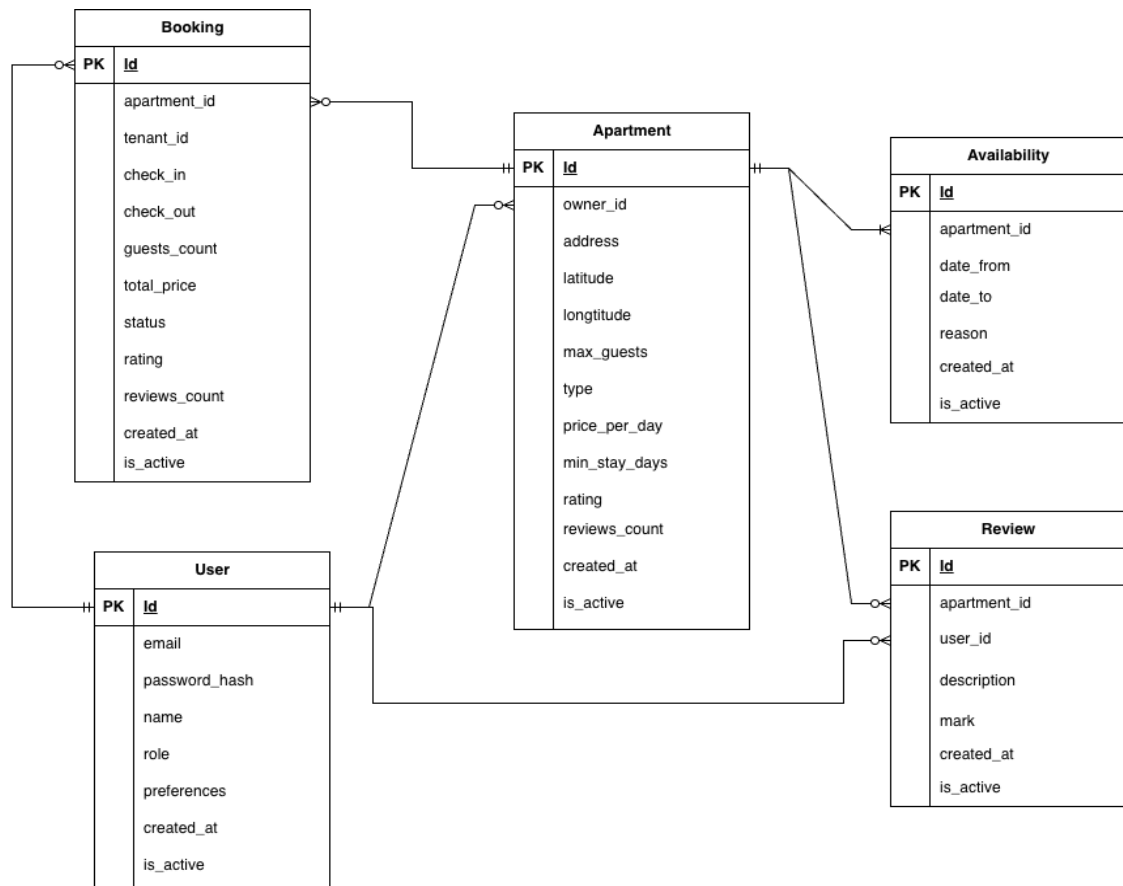


Рисунок 2.2 – Спрощена ER діаграма

Сутність Apartment (апартаменти) є центральною в моделі. Вона зберігає всі характеристики, за якими об'єкт фільтрується та ранжується. До обов'язкових атрибутів належать: ідентифікатор, посилання на орендодавця, адреса у текстовому вигляді, географічні координати (широта і довгота), тип об'єкта, максимальна місткість, ціна за добу, мінімальний строк оренди та статус активності. Набір зручностей зберігається як окрема таблиця зі зв'язком багато-до-багатьох, що дозволяє додавати нові типи зручностей без зміни структури основної таблиці.

Сутність Booking (бронювання) фіксує кожне підтверджене або очікуване резервування. Ключові атрибути: ідентифікатор, посилання на об'єкт та орендаря, дата заїзду, дата виїзду, кількість гостей, підсумкова вартість, статус бронювання та часова мітка створення. Статус може приймати значення: очікує підтвердження, підтверджено, скасовано орендарем, скасовано орендодавцем. Розрізнення причин скасування необхідне для коректного розрахунку повернення коштів відповідно до умов конкретного оголошення.

Сутність Availability (доступність) дозволяє орендодавцю блокувати конкретні дати незалежно від бронювань. Це покриває сценарій, коли власник використовує апартаменти самостійно або тимчасово знімає їх з оренди. Атрибути: посилання на об'єкт, дата початку блокування, дата завершення та необов'язкова причина. При перевірці доступності об'єкта система враховує як наявні бронювання, так і записи в таблиці доступності.

Сутність User (користувач) об'єднує всі ролі системи в одній таблиці з полем ролі: орендар, орендодавець, адміністратор. Один і той самий обліковий запис може виступати як орендарем, так і орендодавцем – це типовий сценарій, коли людина і здає своє житло, і бронює інше. Атрибути: ідентифікатор, email, хешований пароль, ім'я, дата реєстрації та поля для зберігання вподобань, що використовуються при персоналізації ранжування.

Сутність Review (відгук) фіксує оцінку та коментар, залишений користувачем після завершення бронювання. Відгук завжди прив'язаний до конкретного бронювання, що гарантує його достовірність – залишити відгук можна лише за фактом підтвердженого проживання, а не довільно.

2.3.2 Ключові зв'язки між сутностями

Зв'язок між User та Apartment реалізований як один-до-багатьох: один орендодавець може мати кілька об'єктів, кожен об'єкт належить рівно

одному орендодавцю. При видаленні облікового запису орендодавця об'єкти деактивуються, але не видаляються – це зберігає цілісність записів бронювань, що посилаються на ці об'єкти.

Зв'язок між Apartment і Booking є один-до-багатьох: один об'єкт може мати багато бронювань, але кожне бронювання стосується рівно одного об'єкта. Для гарантії відсутності перетинів дат між активними бронюваннями одного об'єкта на рівні бази даних встановлюється обмеження, що перевіряється при кожному INSERT або UPDATE запису бронювання.

Зв'язок між Apartment і Amenity реалізований через проміжну таблицю ApartmentAmenity. Довідник зручностей (Amenity) зберігає стандартизовані назви та категорії: WiFi, паркінг, кухня, пральна машина тощо. Стандартизація критична для роботи фільтрації: якщо зручності зберігаються як довільний текст, надійна фільтрація за ними неможлива.

2.4 Алгоритм відбору та ранжування

2.4.1 Rule-based фільтрація: правила та послідовності перевірок

Логіка фільтрації реалізується як послідовність незалежних перевірок, кожна з яких може виключити об'єкт із подальшого розгляду. Перевірки упорядковані від найефективніших до найменш ефективних з точки зору скорочення кількості кандидатів: перші перевірки мають виключати найбільше об'єктів, щоб мінімізувати обчислювальне навантаження наступних. Така стратегія відповідає принципу раннього відсіювання (early termination), що дозволяє уникнути виконання ресурсоємних перевірок для об'єктів, які вже були виключені на попередніх етапах.

Перша перевірка це доступність дат. Система виконує SQL-запит, що повертає лише ті об'єкти, для яких у вказаному діапазоні дат немає ні активних бронювань, ні записів блокування в таблиці доступності. Ця

перевірка одночасно є найважливішою (бронювання недоступного об'єкта є неприпустимим) та найефективнішою за скороченням вибірки, оскільки при пікових сезонах більшість об'єктів у популярному місті вже заброньовані.

Друга перевірка – відповідність місткості. Об'єкти, максимальна місткість яких менша за вказану кількість гостей, виключаються. Ця перевірка реалізується на рівні SQL-запиту як умова WHERE, що не потребує окремого кроку обробки.

Третя перевірка це ціновий діапазон. Якщо користувач вказав мінімальну або максимальну ціну за добу, об'єкти за межами цього діапазону виключаються. Оскільки ціна за добу може бути динамічною та залежати від конкретних дат, для точної перевірки використовується розрахункова ціна на вказаний період, а не базова ціна з таблиці об'єктів.

Четверта перевірка це наявність обов'язкових зручностей. Якщо користувач позначив певні зручності як обов'язкові, об'єкти, у яких вони відсутні, виключаються через запит до таблиці ApartmentAmenity. Ця перевірка є менш ефективною за скороченням вибірки, ніж попередні, тому виконується останньою серед фільтраційних кроків.

П'ята перевірка це географічна відповідність. Якщо пошук здійснюється за радіусом від вказаної точки, застосовується геопросторова умова фільтрації на основі координат об'єкта. PostgreSQL забезпечує ефективне виконання такого запиту через підтримку геопросторових індексів. Зокрема, використання розширення PostGIS дозволяє оперувати географічними типами даних та виконувати обчислення відстаней з урахуванням кривизни земної поверхні, що забезпечує точніші результати порівняно зі спрощеними евклідовими обчисленнями.

Результатом роботи фільтраційного модуля є набір ідентифікаторів об'єктів, що пройшли всі перевірки. Цей набір може бути порожнім (у такому випадку система повертає відповідь із пустими результатами та переліком умов, що не були виконані жодним об'єктом). Це дає орендарю можливість зрозуміти, які параметри варто послабити.

2.4.2 Алгоритм зваженого ранжування

Ранжування виконується над відфільтрованим набором кандидатів і присвоює кожному об'єкту підсумкову оцінку за формулою зваженої суми нормалізованих значень критеріїв. Нормалізація необхідна, щоб привести критерії з різними одиницями виміру (гривні, кілометри, бали рейтингу) до єдиної шкали від 0 до 1.

До базових критеріїв ранжування належать чотири показники. Перший це відносна ціна: вартість об'єкта порівнюється з середньою ціною аналогічних об'єктів у тому самому районі на ті самі дати. Об'єкт із ціною нижче середньої отримує вищу оцінку за цим критерієм. Другий це рейтинг відгуків (нормалізоване значення середнього рейтингу з поправкою на кількість відгуків). Об'єкт із десятьма відгуками на рейтинг 4.8 ранжується вище, ніж об'єкт із двома відгуками на той самий рейтинг. Третій це відстань до точки пошуку. Четвертий – повнота профілю (частка заповнених атрибутів об'єкта від загального переліку). Об'єкти з детально заповненими картками ранжуються вище, що стимулює орендодавців надавати повну інформацію.

Підсумкова оцінка розраховується за формулою зваженої суми:

$$S = \sum_{c=1}^C w_c \cdot IND_c, \quad (2,1)$$

де IND_c – нормалізоване значення с-го критерію, що обчислюється за формулою:

$$IND_c = \frac{x_c - x_c^{min}}{x_c^{max} - x_c^{min}}, c = \underline{1, C}, \quad (2,2)$$

для критеріїв, де менше значення є кращим (відстань, ціна):

$$INC_c = \frac{x_c^{max} - x_c}{x_c^{max} - x_c^{min}}, \quad (2,3)$$

де w_c – ваговий коефіцієнт c -го критерію, $c = 1, C$, де

$$\sum_{c=1}^C w_c = 1, \quad (2,4)$$

де S – підсумкова оцінка об'єкта, $S \in [0, 1]$.

Результатом роботи модуля ранжування є впорядкований список об'єктів із підсумковими оцінками та розбивкою внеску кожного критерію. Ця розбивка передається клієнтській частині і використовується для відображення пояснення позиції об'єкта. Наприклад: «Об'єкт на 3-му місці: висока оцінка за ціною (+0.38) і повнотою профілю (+0.21), але нижча оцінка за відстанню (−0.12)».

Персоналізація реалізується через зміну коефіцієнтів для конкретного профілю користувача: якщо з попередніх бронювань відомо, що орендар надає перевагу центральному розташуванню і рідко звертає увагу на ціну, вага критерію відстані підвищується, а критерію ціни знижується.

2.4.3 Обробка конфліктних ситуацій при бронюванні

Одночасне бронювання одного об'єкта двома орендарями є задачею, що потребує окремого підходу. Стандартний алгоритм оптимістичного блокування в цьому контексті виглядає наступним чином.

Перший крок це перед відображенням кнопки підтвердження бронювання система виконує повторну перевірку доступності об'єкта на вказані дати. Якщо з моменту пошуку до моменту підтвердження стан змінився, користувачеві відображається актуальна інформація та пропонується обрати інші дати або інший об'єкт.

Другий крок це при підтвердженні бронювання INSERT у таблицю Booking виконується в транзакції, що включає повторну перевірку відсутності конфліктних записів. Якщо між першою і другою перевіркою інший запит встиг зафіксувати бронювання на ті самі дати, транзакція відкочується і орендарю повертається відповідь про недоступність. Ця ситуація є штатною і обробляється на рівні клієнтського інтерфейсу з відповідним повідомленням.

Третій крок – лише після успішної фіксації запису в базі даних система ініціює платіжну транзакцію. Місце не вважається заброньованим до отримання підтвердження від платіжного сервісу. Якщо платіж не пройшов, запис бронювання видаляється або переводиться в статус скасованого, вивільняючи дати для інших орендарів.

2.5 Технічний стек

2.5.1 Серверна частина

Вибір технічного стеку ґрунтується на вимогах, сформульованих у першому розділі, і на характеристиках самих задач, що вирішуються. Для серверної частини обрано .NET із фреймворком ASP.NET Core. ASP.NET Core має вбудовані механізми dependency injection, аутентифікації JWT та middleware-пайплайн для обробки запитів. Для роботи з базою даних використовується Entity Framework Core – ORM, що забезпечує типізований доступ до даних, управління міграціями схеми та підтримку LINQ-запитів. Разом вони утворюють повноцінний інструментарій для побудови RESTful API без потреби у великій кількості сторонніх бібліотек. При виборі технологічного стеку було роглянуто можливість використання інших варіанти. Порівняння можливих варіантів представлено у таблиці 2.1.

Таблиця 2.1 – Переваги та недоліки потенційних технологічних стеків

Потенційний технологічний стек	Переваги	Недоліки	Обрано (так/ні)
C#/.NET, ASP.NET Core	Висока продуктивність і масштабованість; вбудовані механізми dependency injection, аутентифікації та авторизації; підходить для складної логіки та транзакційних операцій	Вищий поріг входу для початківців порівняно з Node.js або Django; розробка може бути більш формалізованою і вимагати більше шаблонного коду;	так
TypeScript/NodeJS	Єдина мова для фронтенду і бекенду; швидкий старт розробки; великий вибір npm-бібліотек; зручний для побудови REST API та real-time функцій;	Слабша строгість архітектури без додаткових обмежень; залежність від сторонніх пакетів; складніше підтримувати велику бізнес-логіку; гірше підходить для, якщо система швидко росте	ні

Продовження таблиці 2.1

Python/Django	Дуже швидка розробка завдяки «batteries included» підходу; вбудована адмін-панель; зручна ORM; зрозуміла структура проєкту; велика кількість бібліотек для аналітики, ML та рекомендаційних модулів	Нижча продуктивність порівняно з .NET у високонавантажених сценаріях; Python слабший у плані суворої типізації та контролю архітектурної дисципліни; для складних асинхронних або високонавантажених API може потребувати додаткових рішень	ні
---------------	---	---	----

Для реалізації геопросторових запитів використовується бібліотека NetTopologySuite у поєднанні з PostGIS – розширенням PostgreSQL для геопросторових даних, що виявилось визначною перевагою для обрання PostgreSQL. Entity Framework Core підтримує NetTopologySuite нативно через відповідний провайдер Npgsql, що дозволяє виконувати пошук об'єктів у радіусі від заданої точки засобами SQL без обчислення відстаней на рівні застосунку.

Для виконання фонових задач використовується Hangfire: надсилання підтверджувальних повідомлень, розрахунок агрегованих показників ранжування та інвалідація кешу. Такі задачі не потребують негайного виконання в рамках запиту і не повинні збільшувати час відповіді API. Hangfire зберігає чергу задач у тій самій базі даних PostgreSQL, що спрощує

інфраструктуру порівняно з рішеннями, що вимагають окремого брокера повідомлень.

2.5.2 Клієнтська частина

Клієнтська частина реалізується на React. Вибір обґрунтований компонентною моделлю, що дозволяє ізолювати логіку відображення кожного елемента інтерфейсу: картка об'єкта, форма пошуку, календар доступності, форма бронювання є незалежними компонентами із власним станом. Це спрощує розробку та тестування інтерфейсу.

Для роботи з картою та відображення об'єктів за географічним розташуванням використовується бібліотека Leaflet із відкритими картографічними тайлами. Це покриває вимогу до пошуку за радіусом від заданої точки та дозволяє відображати результати пошуку одночасно у вигляді списку і на карті.

Стан застосунку управляється через Redux Toolkit, що забезпечує передбачувану логіку оновлення даних і зручне налагодження. Для асинхронних запитів до API використовується RTK Query – компонент Redux Toolkit, що забезпечує кешування відповідей на стороні клієнта та автоматичне скасування застарілих запитів.

2.5.3 Інфраструктура та розгортання

Система розгортається у контейнеризованому середовищі на базі Docker Compose. Кожен компонент (серверна частина, база даних PostgreSQL з PostGIS та pgvector, Redis) запускається у власному контейнері з визначеними залежностями між ними. Така ізоляція унеможливорює конфлікти між версіями бібліотек і спрощує горизонтальне масштабування окремих сервісів. Конфігурація описується у файлі `docker-compose.yml`, що дозволяє відтворити повне середовище однією командою як на локальній

машині розробника, так і на виробничому сервері. Це гарантує відтворюваність середовища розробки та спрощує розгортання на виробничому сервері.

Nginx використовується як зворотний проксі-сервер, що обробляє вхідні HTTP-запити, роздає статичні файли клієнтської частини та перенаправляє API-запити до ASP.NET-застосунку. Завдяки цьому клієнт взаємодіє лише з одним точкою входу, що спрощує налаштування CORS і централізує обробку SSL-сертифікатів. Додатково Nginx виконує кешування відповідей і балансування навантаження, що позитивно впливає на час відгуку системи під пікових навантажень.

2.6 Заплановані результати

Описана архітектура і алгоритмічні рішення є прямою відповіддю на проблеми, виявлені в першому розділі. Непрозорість ранжування вирішується через передачу розбивки оцінок клієнтській частині та їх відображення поряд із результатами. Обмеженість персоналізації вирішується через профіль вподобань і можливість коригування коефіцієнтів ваги для конкретного орендаря. Слабка підтримка складних сценаріїв вирішується через розширювану систему правил фільтрації, де нові умови додаються як окремі перевірки без зміни загальної структури пайплайну. Ризик подвійного бронювання вирішується через транзакційну фіксацію з повторною перевіркою доступності. Таким чином, кожна з виявлених проблем отримує конкретне технічне рішення, що реалізоване на рівні архітектури системи, а не як поверхневе виправлення окремих симптомів. Сукупність цих рішень формує цілісну систему, здатну забезпечити прозорий, персоналізований та надійний процес пошуку і бронювання нерухомості. Загальна схема роботи застосунку зображена на BPMN діаграмі на рисунку 2.3.

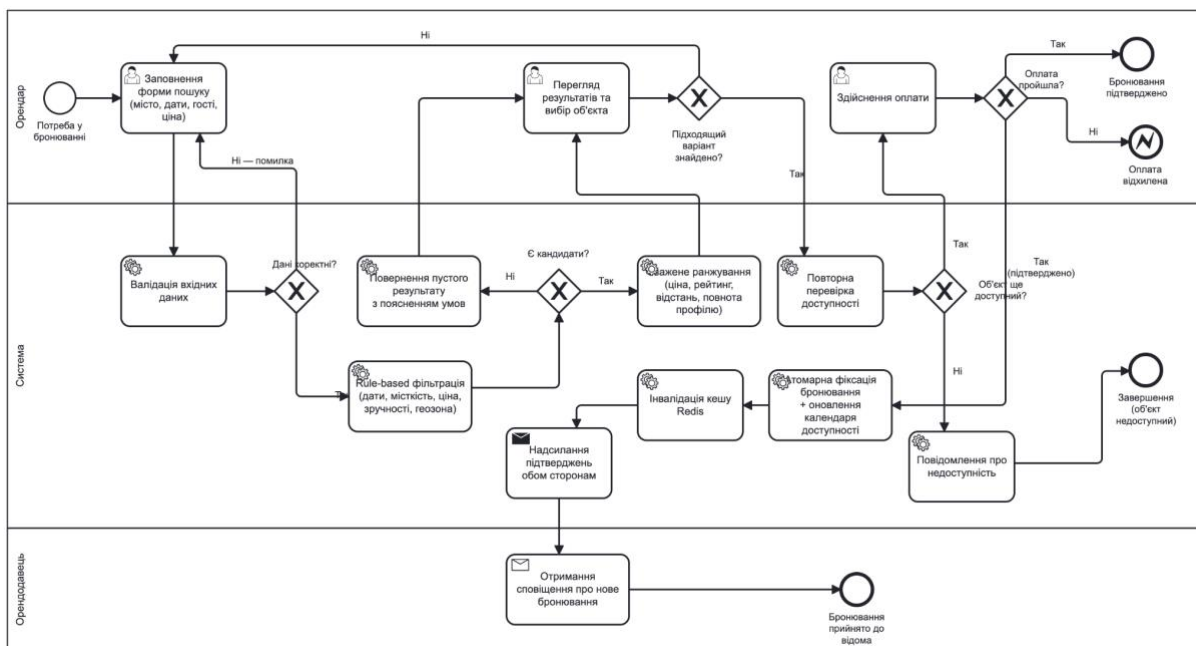


Рисунок 2.3 – BPMN діаграма

Підсумовуючи, було проаналізовано та обґрунтовано технологічний стек системи. На основі порівняльного аналізу обрано ASP.NET Core 8 для серверної частини як платформу з розвинуеною підтримкою асинхронного програмування та інтеграцій із зовнішніми сервісами, PostgreSQL як основну реляційну СУБД із розширеннями PostGIS та pgvector для геопросторових запитів та векторного пошуку відповідно, Redis для кешування та зберігання refresh-токенів, React з TypeScript для клієнтського застосунку. Розглянуто та обґрунтовано вибір архітектурних підходів, зокрема чистої архітектури та CQRS.

Розроблювана система є MVP (minimal viable product), але має свої переваги: кожне рішення щодо алгоритму відбору та ранжування є обґрунтованим. Це дозволяє аналізувати вплив змін у логіці ранжування на якість результатів (задача, яка в комерційних платформах закрыта через непрозорість їх алгоритмів). Саме ця прозорість і відтворюваність логіки відбору створює підґрунтя для наступного кроку – інтеграції методів штучного інтелекту, які здатні не лише автоматизувати ранжування, а й адаптувати його до індивідуальних потреб користувача.

3 ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В СИСТЕМІ БРОНЮВАННЯ АПАРТАМЕНТІВ

3.1 Місце штучного інтелекту в архітектурі системи

Розглядаючи систему бронювання апартаментів як інформаційний посередник між масивом пропозицій та конкретним запитом користувача, стає очевидним, що жодна повністю детерміністична система не здатна повноцінно вирішити задачу релевантного відбору. Rule-based фільтрація та зважене ранжування, описані в другому розділі, вирішують структуровану частину задачі: вони коректно відсіюють неприпустимі варіанти та впорядковують їх за кількісними критеріями. Однак між тим, що користувач вводить у форму пошуку, і тим, що він насправді хоче знайти, існує семантичний розрив, усунення якого потребує інших підходів [6].

Штучний інтелект у контексті даної системи виконує роль не заміни, а доповнення до детерміністичної логіки. Це принципова відмінність від підходу, де AI підміняє ранжування цілком. Замість цього AI-компоненти вбудовані у конкретні точки пайплайну, де вони вирішують задачі, з якими правила та формули справляються гірше: розуміння змісту текстового опису, виявлення прихованих вподобань користувача на основі поведінкових даних та оцінювання якості оголошень. Кожен з цих компонентів функціонує незалежно і вносить свій вимірюваний внесок у підсумковий результат.

Архітектурно AI-компоненти системи поділяються на три функціональні підсистеми.

Перша – це семантичний пошук на основі векторних представлень тексту, що забезпечує розуміння сенсу вільного текстового запиту та опису об'єкта поза межами лексичного збігу.

Друга – підсистема персоналізації ранжування, що адаптує вагові коефіцієнти критеріїв під конкретного користувача на основі його поведінкової історії.

Третя – це автоматична оцінка якості оголошень, що використовує мовну модель для структурованого аналізу повноти та інформативності опису кожного об'єкта і перетворює цей результат на числовий показник, що враховується при ранжуванні.

Усі три підсистеми взаємодіють із центральним пайплайном пошуку, описаним у розділі 2, і не порушують його загальну структуру.

3.2 Семантичний пошук на основі векторних представлень тексту

3.2.1 Обмеження класичного пошуку за ключовими словами

Стандартний підхід до пошуку за текстом базується на лексичному збігу. Тобто, система шукає документи, що містять ті самі слова або їхні форми, що й запит. В умовах короткострокової оренди цей підхід породжує системну проблему: запит і опис об'єкта можуть описувати одне й те саме принципово різними словами. Користувач, який шукає «тихе місце для дистанційної роботи з хорошим освітленням», формулює потребу природньою мовою, тоді як орендодавець у своєму описі може написати «затишна студія з робочим кутком, великими вікнами та спокійним двором». Лексичного перетину між цими двома текстами майже немає, тоді як їхній зміст є практично ідентичним.

Ця проблема відома у теорії інформаційного пошуку як *vocabulary mismatch*. Зазначається, що найкращими термінами для ідентифікації змісту є ті, що мають високу частоту у релевантних документах та низьку загальну частоту у колекції [10], проте такий підхід є фундаментально обмеженим, коли запит і документ описують одне поняття різними словами. Для предметної галузі бронювання апартаментів вона особливо відчутна,

оскільки описи об'єктів пишуться в довільній формі різними авторами з різним словниковим запасом. Жодне розширення словника синонімів або морфологічний аналіз не вирішують задачу повноцінно, оскільки семантичні зв'язки між поняттями є набагато складнішими за прості синонімічні заміни.

3.2.2 Принцип роботи векторних представлень тексту

Сучасний підхід до подолання цього обмеження базується на векторних представленнях тексту (embeddings). Embedding-модель перетворює будь-який текстовий фрагмент на вектор числових значень у багатовимірному просторі, де семантично схожі тексти розташовані поряд, а семантично відмінні далеко один від одного. Ця властивість є ключовою: вона дозволяє вимірювати сенсову схожість між текстами через геометричну відстань між їхніми векторними представленнями, не вдаючись до порівняння окремих слів. Було запропоновано архітектури для обчислення неперервних векторних представлень слів, де було встановлено, що такі вектори здатні відображати тонкі семантичні зв'язки між поняттями [6].

Для вимірювання схожості між векторами застосовується косинусна подібність. На відміну від евклідової відстані, що залежить від довжини векторів, косинусна подібність враховує лише кут між ними, що робить її інваріантною щодо масштабу і більш придатною для порівняння текстів різної довжини. Значення косинусної подібності знаходиться в діапазоні від -1 до 1 , де 1 означає ідентичну семантику, 0 – відсутність смислового зв'язку. На практиці для текстів у предметній галузі бронювання характерні значення в діапазоні 0.6 – 0.9 для семантично споріднених описів.

У системі використовується модель text-embedding-3-small від OpenAI, що генерує вектори розмірністю 1536 вимірів. Вибір цієї моделі обґрунтований балансом між якістю представлень та вартістю генерації.

Порівняно з text-embedding-3-large (розмірність 3072), модель text-embedding-3-small забезпечує достатній рівень семантичної точності для задачі пошуку об'єктів нерухомості. Згідно з документацією OpenAI, модель text-embedding-3-small генерує вектори розмірністю 1536 та забезпечує оптимальний баланс між якістю представлень і вартістю генерації [12].

3.2.3 Технічна реалізація: pgvector та OpenAI Embeddings API

Зберігання та пошук векторних представлень реалізовано засобами розширення pgvector для PostgreSQL [18]. Це розширення додає до стандартного реляційного сховища нативний тип vector, операції обчислення косинусної подібності та евклідової відстані безпосередньо на рівні SQL, а також спеціалізовані індекси для прискорення пошуку найближчих сусідів.

На рівні схеми бази даних таблиця Apartment розширюється стовпцем типу vector(1536), що зберігає embedding опису об'єкта. Цей вектор генерується автоматично щоразу, коли орендодавець створює або оновлює оголошення: фоновий процес (Hangfire-задача) формує текстовий документ із опису, назви та переліку зручностей, надсилає його до OpenAI Embeddings API та зберігає отриманий вектор у відповідний запис. Такий підхід гарантує, що вектор завжди відповідає актуальному стану оголошення, і не потребує від орендодавця жодних додаткових дій.

Для ефективного пошуку найближчих сусідів на стовпці vector встановлюється індекс типу HNSW (Hierarchical Navigable Small Worlds). Цей алгоритм реалізує наближений пошук найближчих сусідів (ANN) з сублінійною складністю, що дозволяє виконувати семантичний пошук серед десятків тисяч об'єктів за десятки мілісекунд. Було запропоновано алгоритм HNSW для наближеного пошуку найближчих сусідів, де складність пошуку масштабується як $O(\log N)$ [8]. Параметри $m=16$ та

ef_construction=64 забезпечують точність відтворення вище 95% відносно повного перебору.

SQL-запит семантичного пошуку виглядає наступним чином: система обчислює косинусну подібність між вектором запиту та вектором кожного активного об'єкта, відсортовує результати за спаданням схожості та повертає перші N кандидатів з їхніми оцінками. Ключовою особливістю є те, що цей запит виконується після rule-based фільтрації: семантичний пошук проводиться лише серед об'єктів, що вже пройшли детерміністичні перевірки доступності, місткості та цінового діапазону. Це суттєво скорочує кількість векторів для порівняння і гарантує, що семантично релевантні, але практично неприйнятні об'єкти не потрапляють до результатів.

3.2.4 Інтеграція семантичного пошуку в пайплайн ранжування

Результат семантичного пошуку вбудовується в загальний механізм зваженого ранжування як додатковий критерій з власним ваговим коефіцієнтом. Косинусна подібність між запитом і описом об'єкта нормалізується до шкали від 0 до 1 у межах поточної вибірки кандидатів і додається до підсумкової оцінки разом із критеріями ціни, рейтингу, відстані та повноти профілю.

Семантичний пошук активується лише тоді, коли пошуковий запит містить текстовий компонент (вільний опис або ключові слова поза структурованими фільтрами). Якщо користувач здійснює пошук виключно через структуровані параметри (місто, дати, кількість гостей, перелік зручностей), семантичний компонент пропускається і ранжування виконується стандартними критеріями. Це дозволяє уникнути обчислення embedding-у для запитів, де він не несе додаткової цінності, і знизити кількість звернень до зовнішнього API.

Окремим сценарієм є ситуація, коли текстовий запит є надто коротким або загальним. Запити типу «квартира» або «центр» не дають достатньо

семантичної інформації для диференціації об'єктів і фактично знижують якість ранжування. Для таких випадків встановлено поріг мінімальної косинусної подібності: об'єкти зі схожістю нижче 0.45 не отримують внеску від семантичного критерію навіть при активному текстовому пошуку.

3.3 Персоналізація ранжування на основі профілю користувача

3.3.1 Модель вподобань користувача

Персоналізація ранжування є природним розвитком базового механізму зважених критеріїв, описаного в підрозділі 2.4.2. Базовий алгоритм присвоює однакові вагові коефіцієнти всім користувачам, що є свідомим спрощенням: насправді різні орендарі мають суттєво відмінні пріоритети. Для одного ціна є вирішальним фактором, для іншого виключно розташування. Персоналізація перетворює ці відмінності на параметри алгоритму.

Модель вподобань зберігається у стовпці `preferences` таблиці `User` у форматі `JSONB`. `JSONB` обрано замість окремої таблиці з міркувань гнучкості, адже структура вподобань може змінюватися з розвитком системи без необхідності міграцій схеми. Вподобання поділяються на дві категорії. Явні вподобання формуються безпосередньо з дій користувача (збережені об'єкти, застосовані фільтри та відгуки після завершення проживання). Неявні вподобання виводяться з аналізу завершених бронювань: цінового діапазону обраних об'єктів, їхнього розташування відносно центру та складу зручностей.

Конкретна структура `JSON`-документа вподобань включає такі поля: `price_sensitivity` (числовий показник від 0 до 1, де вищі значення означають більшу чутливість до ціни); `location_preference` (рядок із закодованим пріоритетом (центр, передмістя, без обмежень)); `preferred_amenities` (перелік ідентифікаторів зручностей, яким надається

перевага); `avg_booking_price` (середня ціна за добу по завершених бронюваннях). Ці поля оновлюються автоматично після кожного завершеного бронювання фоновією Hangfire-задачею, що перераховує агреговані показники на основі останніх п'яти бронювань, тобто більш давня поведінка поступово втрачає вплив.

3.3.2 Адаптація вагових коефіцієнтів

Вагові коефіцієнти критеріїв ранжування адаптуються під конкретного користувача через систему поправок δ , що додаються до базових значень. Базові коефіцієнти визначені для середнього користувача без відомої поведінки і є точкою відліку. Якщо профіль вподобань недоступний або заповнений недостатньо, система повертається до базових коефіцієнтів без модифікації.

Логіка адаптації будується на трьох основних правилах. По-перше, якщо показник `price_sensitivity` перевищує 0.7, вага критерію відносної ціни збільшується на 0.10, а вага критерію повноти профілю зменшується на 0.10, щоб зберегти нормалізацію. По-друге, якщо показник `location_preference` дорівнює «центр», вага критерію відстані збільшується на 0.12 за рахунок симетричного зменшення ваги ціни. По-третє, наявність `preferred_amenities` додає до формули ранжування додатковий критерій відповідності зручностей із вагою 0.08, взятою рівномірно з решти критеріїв. Після застосування всіх поправок система перевіряє виконання умови нормалізації і за необхідності масштабує коефіцієнти, щоб їхня сума залишалася рівною одиниці.

Важливим аспектом є обмеження максимального відхилення від базових коефіцієнтів. Будь-який ваговий коефіцієнт не може бути змінений більш ніж на ± 0.20 відносно базового значення, а також не може опуститися нижче 0.05. Це запобігає деградації ранжування у випадках, коли

поведінкові дані є нетиповими або суперечливими: система залишається функціональною навіть при екстремальних профілях вподобань.

3.3.3 Проблема холодного старту та її вирішення

Персоналізація на основі поведінкових даних стикається з класичною проблемою холодного старту: новий користувач не має жодної історії бронювань, тому система не може визначити його вподобання і змушена використовувати базові коефіцієнти. Хоча це цілком прийнятна поведінка за замовчуванням, вона не використовує можливості персоналізації навіть тоді, коли деяка інформація про користувача вже є.

Для часткового вирішення проблеми холодного старту системою передбачено необов'язковий сценарій первинного опитування при реєстрації. Після завершення реєстрації користувачу пропонується відповісти на три питання щодо типового сценарію використання платформи (ділові поїздки / відпочинок / сімейний відпочинок), пріоритету між ціною та розташуванням, а також частоти поїздок. На основі відповідей система встановлює початкові значення явних вподобань, не чекаючи накопичення поведінкових даних та опираючись на досвід користувачів із подібним профілем. Опитування є необов'язковим і може бути пропущено, однак у такому разі профіль вподобань залишається порожнім до першого завершення бронювання.

Після першого бронювання система отримує перші неявні дані: ціну, розташування та зручності обраного об'єкта. В разі, якщо початкове опитування було пропущено, цього вже достатньо для початкового заповнення профілю. Дослідження демонструють проблему холодного старту і зазначають, що рекомендаційні системи працюють значно гірше для нових користувачів [14]. Успіх колаборативної рекомендаційної системи залежить від наявності критичної маси користувачів [15]. Це означає, що повноцінна персоналізація активується для користувачів, які скористались

платформою кілька разів, тобто саме та аудиторія, утримання якої є найбільш цінним з бізнес-перспективи.

3.4 Автоматична оцінка якості оголошень

3.4.1 Вплив якості опису на ефективність системи

Якість опису оголошення безпосередньо впливає на ефективність усіх інших компонентів системи. Семантичний пошук дає гірші результати для об'єктів із коротким або неінформативним описом, оскільки embedding, побудований на мінімальному тексті, не відображає реальних характеристик об'єкта. Ранжування за критерієм повноти профілю, описане в підрозділі 2.4.2, враховує лише структуровані атрибути (заповненість полів). Воно не здатне оцінити змістовну якість текстового опису, бо два оголошення з однаково заповненими полями можуть мати принципово різну інформативність.

Задача, що вирішується цим компонентом, полягає у генерації числової оцінки якості текстового опису кожного оголошення на основі аналізу його змісту. Ця оцінка відображає, наскільки опис допомагає потенційному орендарю прийняти обґрунтоване рішення, і враховується при ранжуванні як складова критерію повноти профілю.

3.4.2 Реалізація оцінювання засобами мовної моделі

Оцінювання якості опису виконується через структурований промпт до мовної моделі gpt-4o. Для кожного оголошення формується запит, що містить повний текст опису та інструкцію повернути JSON-об'єкт з числовими оцінками за чотирма критеріями: інформативність (наскільки конкретно описані характеристики простору), унікальність, практична корисність (чи містить опис деталі, що впливають на рішення орендаря) та

мовна якість (зв'язність, грамотність, читабельність). Кожен критерій оцінюється за шкалою від 0 до 1 з кроком 0.1. Приклад поверненого результату схематично зображено на рисунку 3.1.

informativeness	0.4
uniqueness	0.6
practicalUsefulness	0.4
languageQuality	0.9

Рисунок 3.1 – JSON схема, що повертається при оцінюванні засобами мовної моделі

Підсумкова оцінка якості опису є зваженим середнім чотирьох критеріїв з вагами 0.35, 0.25, 0.30 та 0.10 відповідно. Домінування інформативності та практичної корисності в загальній оцінці відображає пріоритет, бо орендарю важливіше отримати конкретну інформацію, ніж читати красиво написаний, але беззмістовний текст. Підсумкова оцінка нормалізується і зберігається у таблиці Apartment.

Для мінімізації затримки та вартості оцінювання процес виконується асинхронно у фоновій Hangfire-задачі. Оцінка розраховується одноразово при публікації оголошення та повторно при будь-якій зміні текстового опису. Вартість одного запиту до API для аналізу середнього опису складає близько 0.002–0.004 USD, що є прийнятним для однократного розрахунку при публікації.

3.4.3 Вплив оцінки якості на ранжування та поведінку орендодавців

Оцінка якості опису замінює попереднє визначення критерію повноти профілю в алгоритмі ранжування. Раніше цей критерій обчислювався як

відношення заповнених структурованих полів до загальної кількості полів. Тепер він є зваженим середнім структурної повноти (частка заповнених полів, вага 0.4) та якості опису (`description_quality_score`, вага 0.6). Така комбінація враховує як наявність даних, так і їхню змістовну цінність.

Практичний ефект цього компонента є двоїстим. З одного боку, об'єкти з детальними та інформативними описами отримують вищу позицію в результатах пошуку, що безпосередньо зацікавляє орендодавців у підвищенні якості своїх оголошень. З іншого боку, семантичний пошук стає точнішим: `embedding`, побудований на якісному описі, краще представляє реальні характеристики об'єкта і частіше збігається з семантикою запитів відповідних орендарів. Таким чином, оцінка якості опису позитивно впливає на два незалежні компоненти системи одночасно.

3.5 Рекомендації схожих об'єктів

Векторні представлення, що генеруються для семантичного пошуку, слугують основою ще для одного AI-компонента – рекомендацій схожих об'єктів. Коли орендар переглядає сторінку конкретного апартаменту, система відображає підбірку з чотирьох-шести схожих пропозицій, що можуть зацікавити його як альтернатива або для порівняння.

Схожість між об'єктами визначається як косинусна подібність між їхніми `embedding`-ами, доповнена структурними обмеженнями: рекомендовані об'єкти мають перебувати у тому самому місті, мати місткість у межах ± 2 від переглянутого та бути активними. Такий гібридний підхід (семантична подібність при дотриманні структурних обмежень) забезпечує релевантність рекомендацій без ризику запропонувати принципово непридатні варіанти.

З точки зору продуктивності рекомендації є попередньо розрахованими: при індексуванні нового або оновленого оголошення Hangfire-задача перераховує список найближчих сусідів та зберігає його у

спеціальній таблиці ApartmentSimilarity. Сторінка об'єкта отримує рекомендації простим SELECT за ідентифікатором, без звернення до векторного індексу в режимі реального часу. Це зводить затримку відображення рекомендацій до нуля відносно часу відповіді основного запиту.

3.6 Оцінювання адекватності системи штучного інтелекту

Оцінка адекватності AI-модуля передбачає перевірку відповідності обраних методів природі задач, що вирішуються, та обґрунтованості прийнятих проєктних рішень у контексті предметної галузі з урахуванням як переваг, так і обмежень кожного компонента.

Застосування векторних представлень для подолання vocabulary mismatch є природним вибором для предметної галузі короткострокової оренди. Описи апартаментів є достатньо довгими та змістовними текстами, щоб сформувати інформативний вектор, а косинусна подібність як метрика є інваріантною до довжини опису, що усуває систематичне зміщення на користь коротших текстів. Індекс HNSW із задекларованою точністю вище 95% є достатнім для задачі ранжування, де незначне зміщення у складі кандидатів не чинить критичного впливу на кінцевий результат. Разом з тим підхід має суттєве обмеження: якість семантичного пошуку повністю визначається якістю вхідного тексту. Embedding відображає зміст того, що написано в описі, а не реальні характеристики об'єкта, тому неточний або відсутній опис не компенсується моделлю. Додатковим обмеженням є залежність від зовнішнього API при генерації векторів: недоступність сервісу унеможлиблює індексацію нових оголошень, хоча асинхронна архітектура мінімізує наслідки цього ризику для кінцевого користувача.

Підхід до персоналізації на основі явних корекцій вагових коефіцієнтів δ є адекватним для поточного масштабу системи. Його ключовою перевагою є незалежність від критичної маси користувачів:

колаборативна фільтрація або матрична факторизація потребують щільної матриці взаємодій, яка не формується на початковому етапі розгортання, тоді як обраний підхід коректно функціонує від першого бронювання. Обмеження максимального відхилення ± 0.20 від базових коефіцієнтів ефективно запобігає деградації ранжування при нетипових поведінкових даних. Проте підхід має очевидні недоліки. Вікно агрегації з п'яти останніх бронювань є евристичним рішенням без теоретичного обґрунтування оптимальності: для користувачів із нерегулярною поведінкою, наприклад тих, хто чергує ділові та відпочинкові поїздки, агреговані показники стабільно не відображатимуть актуальних вподобань. Модель також не враховує контекст конкретного пошукового сеансу: один і той самий користувач може мати суттєво різні потреби залежно від мети поїздки, тоді як система застосовує єдиний скоригований профіль незалежно від контексту.

Застосування мовної моделі для структурованого аналізу текстового опису є адекватним рішенням для задачі, що не має прийнятного детерміністичного аналога. Перевагою підходу є здатність відрізнити змістовний опис від беззмістовного тексту аналогічного обсягу – завдання, з яким жодна евристика на основі кількості символів або наявності ключових слів не справляється. Чотирикритеріальна схема з поверненням структурованого JSON забезпечує стабільний парсинг і прозору інтерпретацію результату. Проте стохастична природа генерації означає, що два послідовні запити з ідентичним текстом можуть повертати дещо відмінні числові значення, що є потенційним джерелом нестабільності оцінки. Для задачі ранжування, де `description_quality_score` є одним із кількох зважених критеріїв, такий рівень варіативності є прийнятним, однак не усуненим. Суттєвим обмеженням є також відсутність верифікації: відповідність оцінок моделі реальному сприйняттю орендарів не підтверджена емпірично, а критерії оцінювання відображають проєктні припущення, а не результати користувацьких досліджень.

Попередній розрахунок векторних сусідств із збереженням у таблиці ApartmentSimilarity є архітектурно виправданим рішенням: характеристики об'єктів змінюються значно рідше, ніж вони переглядаються, тому актуальність кешованих результатів є достатньою при регулярному перерахунку. Гібридний підхід це семантична подібність при дотриманні структурних обмежень за містом та місткістю. Він коректно балансує між релевантністю та практичною придатністю рекомендацій. Недоліком є статичність рекомендацій у межах між перерахунками: щойно опублікований об'єкт не з'явиться у рекомендаціях до завершення фонові задачі, а для активно оновлюваного каталогу це вікно неактуальності може бути відчутним.

Загалом, обрані методи є адекватними для поставлених задач у межах визначеного масштабу системи. Кожен компонент вирішує задачу, для якої він є природним інструментом, а виявлені недоліки є або прийнятними компромісами для поточного етапу розробки, або усувними в наступних ітераціях без зміни загальної архітектури. Необхідно зазначити, що наведена оцінка є аналітичною, а не емпіричною: кількісна верифікація якості семантичного пошуку потребує розміченої тестової вибірки, оцінка персоналізації – A/B тестування на реальній аудиторії, а відповідність оцінок якості опису людській експертній оцінці потребує окремого валідаційного дослідження, що становлять природні напрями подальшого розвитку системи.

3.7 Обмеження AI-компонентів та заходи їх мінімізації

Використання зовнішніх AI-сервісів, зокрема OpenAI API, вносить у систему залежність від третьої сторони. Недоступність API означає неможливість генерації нових embedding-ів при публікації оголошень та нових оцінок якості. Для мінімізації наслідків цього ризику обидва процеси є асинхронними і не блокують публікацію оголошення: об'єкт стає

доступним для перегляду одразу, а AI-обробка виконується у фоні. Якщо задача завершилася помилкою, планувальник автоматично повторює її з експоненціальним відступом.

Інше обмеження пов'язане з якістю вхідних даних для семантичного пошуку. Embedding відображає зміст того, що написано в описі, а не реальні характеристики об'єкта. Якщо опис є неточним або відсутнім, семантичний пошук не компенсує цього і може повертати нерелевантні результати. Це обмеження частково нейтралізується компонентом оцінки якості опису. Об'єкти з низьким `description_quality_score` отримують менший внесок від семантичного критерію у підсумковій оцінці ранжування.

Персоналізація ранжування обмежена обсягом поведінкових даних. При малій кількості бронювань зміщення вагових коефіцієнтів може бути нестабільним і відображати окремі нетипові випадки, а не стійкі вподобання. Система обмежує максимальне відхилення від базових коефіцієнтів, як описано в підрозділі 3.3.2, що запобігає деградації якості ранжування навіть при недостатній кількості поведінкових даних. Стабілізація відбувається природно у міру накопичення бронювань.

Загалом описані AI-компоненти проєктуються за принципом *graceful degradation*. Кожен із них при відмові або недоступності повертається до базової поведінки без порушення роботи решти системи. Семантичний пошук без `embedding` повертається до повнотекстового пошуку PostgreSQL. Персоналізація без профілю використовує базові коефіцієнти. Оцінка якості без відповіді API залишає попереднє збережене значення або встановлює нейтральне значення 0.5 для нових оголошень. Така архітектурна стійкість є запорукою того, що впровадження AI-компонентів не знижує, а підвищує загальну надійність системи.

Отже, було спроектовано архітектуру AI-компонентів системи. Розроблено багатокритеріальний зважений алгоритм ранжування, що враховує ціну, байєсівський рейтинг, географічну відстань та повноту профілю оголошення. Для алгоритму обґрунтовано базові вагові

коефіцієнти та механізм їх персоналізації на основі збережених уподобань автентифікованого користувача. Спроектовано рішення семантичного пошуку на основі векторних представлень OpenAI та косинусної подібності pgvector, де кінцева оцінка є зваженою комбінацією семантичного та структурного компонентів. Детально описано компонент автоматичної оцінки якості текстових описів оголошень, реалізований через звернення до OpenAI API, та систему рекомендацій схожих об'єктів на основі попередньо розрахованих векторних сусідств.

Враховуючи наявність конкретних проєктних рішень, наступним кроком виступає практична реалізація мінімально життєздатної версії описаної системи, яка буде включати більшість описаних пунктів.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ БРОНЮВАННЯ АПАРТАМЕНТІВ

4.1 Структура проєкту та архітектура рішення на рівні коду

Реалізація системи ґрунтується на принципах чистої архітектури (Clean Architecture). Цей підхід формалізовано як архітектуру з чітко визначеними напрямками залежностей, де внутрішні шари не залежать від зовнішніх. Проєкт містить два основні компоненти: серверний застосунок на платформі ASP.NET Core 8 та клієнтський застосунок на базі React з TypeScript, а також окремий тестовий проєкт ApartmentBooking.Tests. Серверна частина організована у вигляді єдиного проєкту ApartmentBooking.API із внутрішньою структурою директорій, що відтворює архітектурні шари: Domain, Application, Infrastructure та Controllers. Загальний вигляд структури проєкту в IDE зображено на рисунку 4.1.

Шар Domain містить виключно доменні сутності та переліки (enums) без жодних зовнішніх залежностей. Це забезпечує повну незалежність бізнес-моделі від деталей технічної реалізації – бази даних, фреймворків чи зовнішніх сервісів. Шар Application визначає команди, запити, DTO (Data Transfer Objects), інтерфейси сервісів та валідатори. Тут реалізовано паттерн CQRS за допомогою бібліотеки MediatR, де кожна операція є окремим класом-запитом з відповідним класом-обробником в одному файлі. Шар Infrastructure містить конкретні реалізації інтерфейсів, визначених в Application: контекст Entity Framework Core, сервіси автентифікації, сервіс генерації векторних представлень, реалізацію конвеєра пошуку та сервіс ранжування. Шар Controllers є найтоншим – контролери приймають HTTP-запити, делегують виконання до MediatR або спеціалізованих сервісів та повертають відповідь.

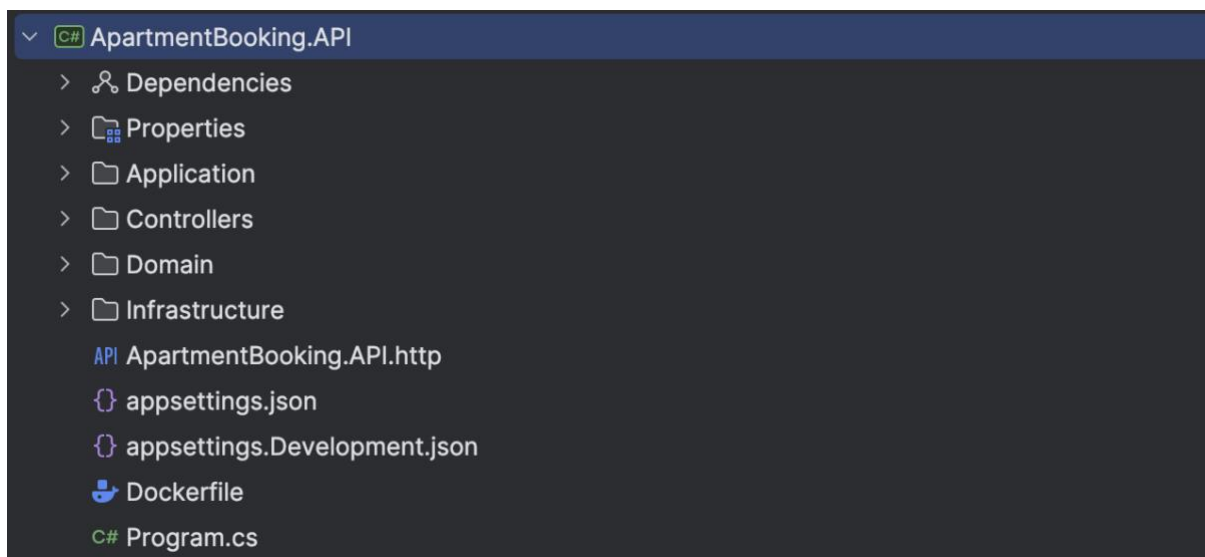


Рисунок 4.1 – структура проєкту в IDE

Для конвеєра валідації використовується бібліотека `FluentValidation`, яка інтегрується в pipeline `MediatR` через поведінку `ValidationBehavior<TRequest, TResponse>`. Це забезпечує автоматичну валідацію будь-якого запиту перед його виконанням без необхідності дублювати валідаційну логіку в обробниках. Якщо валідація не пройдена, pipeline кидає виключення `ValidationException`, яке глобальний обробник помилок перетворює на HTTP-відповідь з кодом 422 Unprocessable Entity у форматі RFC 7807 Problem Details. Обробник також перетворює `NotFoundException` на HTTP 404, `ForbiddenException` – на HTTP 403, `ConflictException` – на HTTP 409, що забезпечує семантично коректні статус-коди для всіх сценаріїв помилок.

Реєстрація залежностей виконується у файлі `Program.cs`, де послідовно налаштовуються: підключення до PostgreSQL через `NpgsqlDataSourceBuilder` із просторовими розширеннями, підключення до Redis як синглтону `ICnectionMultiplexer`, сервіси автентифікації, Hangfire із бекендом на PostgreSQL, конвеєр пошуку з десяти фільтрів у строго визначеному порядку, `MediatR` із вбудованою валідаційною поведінкою та `FluentValidation`. Swagger UI налаштований з підтримкою JWT-авторизації

безпосередньо в інтерфейсі документації, що спрощує ручне тестування захищених ендпоінтів.

4.2 Доменна модель та структура бази даних

Доменна модель включає вісім сутностей: User, Apartment, Booking, Review, Amenity, ApartmentAmenity, Availability та ApartmentPhoto. Кожна сутність відповідає окремому бізнес-об'єкту предметної області та має чітко визначену зону відповідальності, що спрощує подальше розширення моделі без порушення існуючих зв'язків. Проектування бази даних проводилось згідно практик моділювання і оцінки ступеню нормалізації. База даних повністю відповідає вимогам другої нормальної форми: кожен неключовий атрибут функціонально залежить від цілого первинного ключа, часткових залежностей не виявлено. Єдина таблиця зі складеним ключем – ApartmentAmenity. Вони не містить додаткових атрибутів, тому друга нормальна форма виконується.

Повна відповідність третій нормальній формі свідомо не дотримана в одному місці: поля Rating та ReviewsCount таблиці Apartments є похідними від даних таблиці Reviews і, строго кажучи, утворюють транзитивну залежність. Однак їх винесення в окрему таблицю або обчислення через агрегатний запит при кожному зверненні до списку апартаментів було б неприйнятним з точки зору продуктивності, оскільки ці поля використовуються у критичному шляху пошуку та ранжування. Обрана денормалізація є типовою практикою для подібних систем (значення оновлюються інкрементально при додаванні або видаленні відгуку засобами обробника доменної події, що гарантує їх узгодженість із таблицею Reviews). Таке рішення демонструє, що нормалізація є інструментом, а не самоціллю – її застосування має бути збалансоване з вимогами до продуктивності конкретної системи. На рисунку 4.2 зображена ER-діаграма. На рисунку 4.2 зображена ER-діаграма бази даних, що було реалізовано.

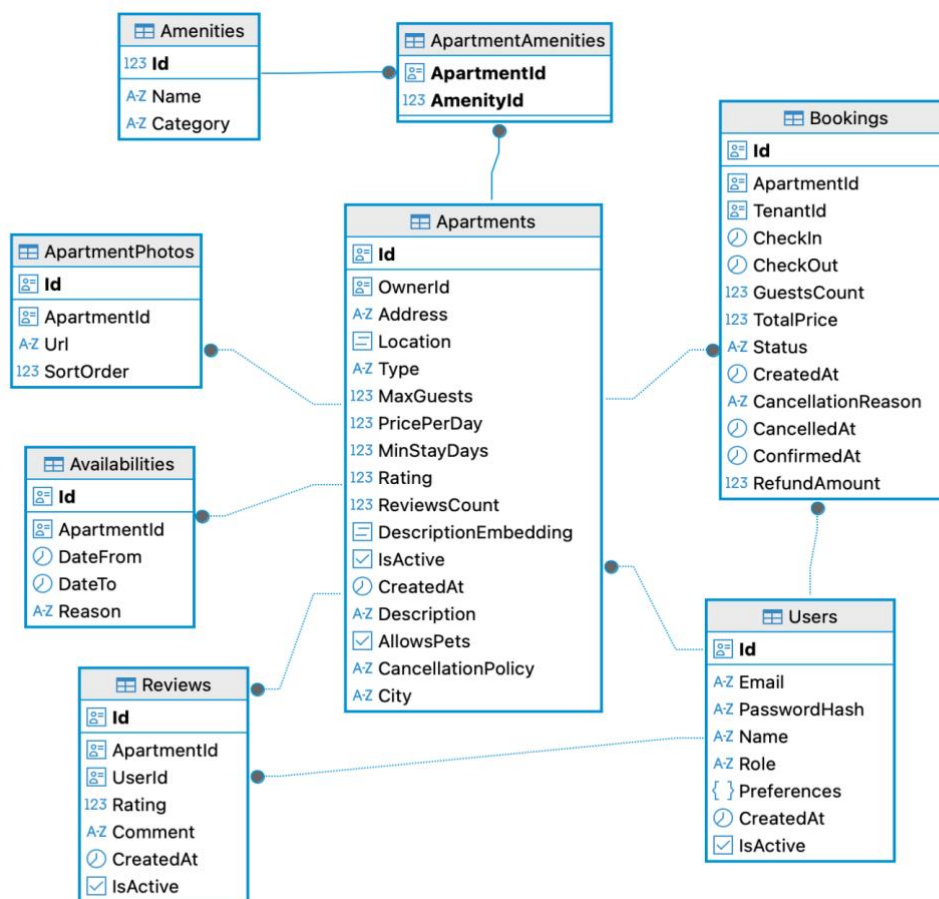


Рисунок 4.2 – ER-діаграма бази даних

Клас Apartment є центральним об'єктом системи та містить два поля, що вимагають спеціальних PostgreSQL-розширень. Поле Location типу Point (бібліотека NetTopologySuite) зберігає геопросторові координати у форматі WGS-84 (SRID 4326) та потребує розширення postgis. Поле DescriptionEmbedding типу Vector (бібліотека Pgvector) зберігає 1536-вимірний вектор тексту опису та потребує розширення vector. Клас User містить поле Preferences типу JsonDocument для зберігання персоналізованих вагових коефіцієнтів ранжування у стовпці jsonb.

Контекст ApplicationDbContext оголошує три PostgreSQL-розширення безпосередньо в методі OnModelCreating:

```

modelBuilder.HasPostgresExtension("postgis");
modelBuilder.HasPostgresExtension("vector");
modelBuilder.HasPostgresExtension("btree_gist");

```

Це змушує Entity Framework Core генерувати відповідні оператори CREATE EXTENSION IF NOT EXISTS у міграціях. Розширення btree_gist є обов'язковим для роботи обмеження виключення (exclusion constraint) на таблиці бронювань, яке запобігає перекриттю дат на рівні самої бази даних як додатковий захисний рівень поверх перевірки у коді. Конфігурації мапінгу сутностей зосереджені в окремих класах-конфігураторах, що реалізують IEntityTypeConfiguration<T>, і завантажуються одним викликом ApplyConfigurationsFromAssembly.

Клас Booking визначає поля CheckIn та CheckOut як DateOnly – тип .NET 6+, що безпосередньо відповідає типу date PostgreSQL без часового компонента. Поле TotalPrice обчислюється при створенні бронювання та зберігається явно, що убезпечує від зміни суми при можливому коригуванні ціни апартаменту в майбутньому. Статус бронювання визначається переліком BookingStatus із чотирма значеннями: Pending, Confirmed, CancelledByTenant та CancelledByLandlord. Таке розрізнення сторони, що скасувала бронювання, є важливим для коректного обчислення суми повернення коштів.

Для підключення до бази даних використовується NpgsqlDataSourceBuilder із явною реєстрацією обох просторових розширень через методи UseNetTopologySuite та UseVector. Рядок підключення передається через змінні оточення в контейнерному середовищі, що унеможливорює потрапляння реальних облікових даних у репозиторій.

4.3 Реалізації автентифікації та управління сесіями

Система автентифікації реалізована на основі JWT Bearer-токенів з ротацією refresh-токенів. AuthController надає чотири HTTP-кінцевих точки: реєстрацію (POST /api/auth/register), вхід (POST /api/auth/login), оновлення токена (POST /api/auth/refresh) та отримання профілю поточного користувача (GET /api/auth/me). При реєстрації пароль хешується

бібліотекою BCrypt.Net із work factor 12, що забезпечує достатній рівень стійкості проти атак перебором при прийнятному часі обчислення. Електронна адреса нормалізується до нижнього регістру перед збереженням, а унікальність перевіряється до спроби вставки, щоб уникнути виключення на рівні бази даних. На рисунку 4.3 зображено розшифрований JWT токен та Claims.

The image shows a JWT token decoder interface. On the left, a long JWT token is displayed in a monospace font. On the right, the decoded payload is shown in a JSON format. Below the JSON, there is a 'Decoded Payload' section with a 'Claims Breakdown' table. The table lists various claims and their values, along with brief descriptions and links to learn more.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIwOWM1OTYwZi03ZWQ2LTQ2NjctYjE0NS00MTM2MjNhODA0MmUiLCJlbwFpbCI6InRlc3RAZ2Z1hWwYyZ9tIiwicm9sZSI6ImlbmFudCIsImp0aSI6IjQ3ZDgwNmEzLTUwMjMtNDA2Yi01ZDU3LWU4YzlkY2JlM2YlLCJleHAiOiE3ODAzMDMxODksImZscyI6IkwYXJ0bWVudEJvb2tpbmdBUEkiLCJhdWQiOiJlbnRlc29raw5nQ2xpZW50IiwiaWF0Ij0iLn0.WwDzYUvW6r9AqUWXHrhLw6FDNR3Nfr9098BxZSHb9AY
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

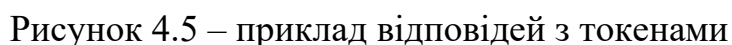
Decoded Payload

JSON Claims Breakdown

sub	09c5960f-7ed6-4667-b145-413623a8042e	The subject of the JWT (the user). Learn more
email	test@gmail.com	
role	Tenant	
jti	47d806a3-5063-406b-5d57-e8c9dcb3f	The unique identifier for the JWT. Learn more
exp	1780603189 (Thu Jun 04 2026 21:59:49 GMT+0200 (Central European Summer Time))	The expiration time on or after which the JWT MUST NOT be accepted for processing. Learn more
This value must be a <code>NumericDate</code> type, representing seconds.		
iss	ApartmentBookingAPI	The issuer of the JWT. Learn more
aud	ApartmentBookingClient	The recipients that the JWT is intended for. Learn more

Рисунок 4.3 – розшифрований JWT токен з Claims

Сервіс TokenService генерує access-токени з вбудованими клеймами ідентифікатора користувача, електронної адреси та ролі. Термін дії access-токена налаштовується через параметр ExpiryMinutes у секції JWT конфігурації. Refresh-токени є криптографічно стійкими рядками, що зберігаються в Redis із обмеженим часом існування. При кожному успішному оновленні старий refresh-токен відкликається до видачі нового, що реалізує схему ротації та запобігає повторному використанню скомпрометованого токена. RefreshTokenService зберігає токени у Redis із форматом ключа refresh: {token}, де значенням є ідентифікатор відповідного користувача. Створені ендпоінти візуально продемонстровані за допомогою



Авторизація в системі ґрунтується на ролях, які декодуються безпосередньо з JWT-клейму «role». Кожному користувачу при реєстрації

призначається одна з трьох ролей: Tenant (орендар), Landlord (орендодавець) або Admin. Спроба зареєструватися з роллю Admin відхиляється на рівні контролера, оскільки адміністративні акаунти можуть бути створені лише безпосередньо через базу даних або скрипт початкового заповнення DataSeeder. Захищені маршрути позначаються атрибутом `[Authorize]` або `[Authorize(Roles = "Landlord")]`, що делегує перевірку фреймворку ASP.NET Core.

4.4 Конвеєр пошуку: реалізація ланцюга фільтрів

Конвеєр пошуку апартаментів побудований за паттерном ланцюга відповідальності (Chain of Responsibility) з використанням інтерфейсу `ISearchFilter`. Кожен фільтр приймає `IQueryable<Apartment>` та об'єкт `SearchQuery`, виконує композицію LINQ-виразів і повертає модифікований `IQueryable`. Оскільки LINQ-вирази є ледачими (lazy), усі фільтри компонуються до єдиного SQL-запиту, який виконується лише один раз при матеріалізації результатів у `SearchPipeline.ExecuteAsync`. Таким чином, незалежно від кількості активних фільтрів, до бази даних надсилається рівно один запит.

Порядок виконання фільтрів визначається порядком реєстрації в контейнері залежностей у файлі `Program.cs`: `ActiveApartmentsFilter` відкидає неактивні оголошення (`IsActive = false`), `AvailabilityFilter` виключає апартаменти, недоступні за заданими датами, `CapacityFilter` перевіряє відповідність місткості кількості гостей, `PriceFilter` застосовує цінові обмеження, `AmenitiesFilter` перевіряє наявність обов'язкових зручностей, `GeographicFilter` фільтрує за містом або географічним радіусом, а наступні чотири фільтри – `MinRatingFilter`, `PetsFilter`, `MinStayFilter` та `CancellationPolicyFilter` – застосовують факультативні критерії.

`AvailabilityFilter` перевіряє два незалежних джерела недоступності: активні бронювання зі статусом `Pending` або `Confirmed` та явно заблоковані

дати від орендодавця (таблиця Availabilities). Умова перекриття дат реалізована через класичну формулу для напіввідкритих інтервалів $[A, B)$ та $[C, D)$: збігаються тоді і лише тоді, коли $A < D$ та $B > C$. Код, що реалізує цю логіку, представлений в лістингу 4.1.

Лістинг 4.1 – Програмний код реалізації LINQ запиту перевірки доступності

```
query = query.Where(a => !a.Bookings.Any(b =>
    (b.Status == BookingStatus.Confirmed ||
     b.Status == BookingStatus.Pending) &&
    b.CheckIn < checkOut &&
    b.CheckOut > checkIn));
```

Ця умова трансліюється LINQ-провайдером PostgreSQL в ефективний SQL із використанням часткового індексу на стовпцях CheckIn та CheckOut для активних бронювань.

GeographicFilter підтримує два режими географічного обмеження: пошук за назвою міста через PostgreSQL-оператор ILIKE (виклик EF.Functions.ILike) та пошук у радіусі через функцію ST_DWithin розширення PostGIS. У реалізації використовується метод IsWithinDistance бібліотеки NetTopologySuite, який ORM-провайдер трансліює в SQL-виклик ST_DWithin. Оскільки стовпець Location зберігається у типі geometry (не geography), відстань задається в градусах із перетворенням кілометрів через коефіцієнт 111,195 км/градус – достатня точність для пошуку в масштабах міста.

AmenitiesFilter реалізує вимогу одночасної наявності кількох зручностей через генерацію окремого підзапиту EXISTS для кожного ідентифікатора зручності. Такий підхід відрізняється від підходу з JOIN, де кожна зручність розширювала б кількість рядків результату, – замість цього кожен EXISTS незалежно використовує складений індекс на таблиці ApartmentAmenities.

Після виконання конвеєра SearchPipeline проєктує результати у список об'єктів ApartmentCandidate. Коли передано географічний центр, відстань від кожного апартаменту до центру обчислюється на стороні бази даних через ST_Distance і повертається разом із рештою полів, уникаючи повторного обчислення у шарі ранжування. Приклад запиту та відповіді візуально зображені на рисунку 4.6.

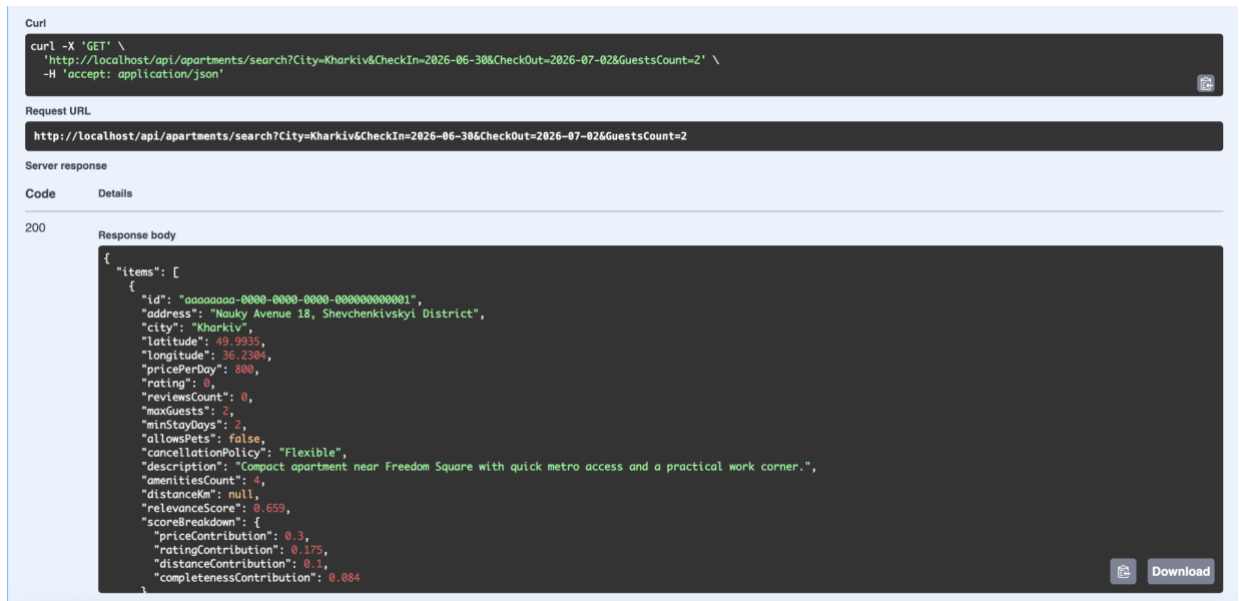


Рисунок 4.6 – Swagger UI POST /api/search/apartments/ranked з тілом запиту та відповіддю

4.5 Алгоритм ранжування: конкретна реалізація

RankingService реалізує зважену багатокритеріальну оцінку з чотирьох компонентів: ціна, рейтинг, відстань та повнота профілю. Усі чотири компоненти нормалізовані до діапазону [0, 1] методом мін-макс нормалізації для забезпечення порівнянності між критеріями з різними одиницями виміру. Базові вагові коефіцієнти визначені константою RankingWeights.Default: ціна – 0.30, рейтинг – 0.35, відстань – 0.20, повнота – 0.15. Якщо автентифікований користувач має збережені вагові

уподобання у полі Preferences, метод RankingWeights.FromUserPreferences зчитує їх, замінює відповідні базові значення та нормалізує суму до одиниці:

```
var sum = price + rating + distance + completeness;
return new RankingWeights(price/sum,
rating/sum, distance/sum, completeness/sum);
```

Нормалізація ціни та відстані є зворотною (inverse): менша ціна і менша відстань відповідають вищому балу. Це реалізовано функцією $\text{SafeNormalizeInc}(x, \min, \max) = (\max - x) / (\max - \min)$. Нормалізація рейтингу є прямою. Перед нормалізацією сирий рейтинг замінюється байєсівською оцінкою за формулою:

$$\text{BayesianRating} = \frac{(\text{rating} \times v + \text{globalAvg} \times M)}{(v + M)}, \quad (4.1)$$

де v – кількість відгуків апартаменту;

globalAvg – середній рейтинг по всіх кандидатах;

$M = 10$ – гіперпараметр, що регулює силу вирівнювання.

Такий підхід запобігає нечесній перевазі апартаментів із одним відгуком з оцінкою 5.0 над об'єктами із сотнею об'єктивних відгуків: при $M = 10$ об'єкт з одним відгуком 5.0 отримає байєсівський рейтинг, що практично дорівнює загальному середньому.

Оцінка повноти профілю ScoreCompleteness нараховує бали за заповнені поля: по одному за кожне з дев'яти структурованих полів (адреса, місто, координати, тип, місткість, ціна, мінімальна тривалість, дозвіл для тварин, політика скасування) та ще один бал за наявність текстового опису. Додатково нараховується до 10 балів пропорційно кількості вказаних зручностей (ліміт $\text{cap} = 10$). Загальна сума нормалізується до $[0, 1]$ діленням на максимально можливу кількість балів (25). Функції SafeNormalize та SafeNormalizeInc реалізують захист від ділення на нуль: якщо всі кандидати мають однакове значення критерію, функція повертає нейтральне значення 0.5, не вводячи штучного розрізнення.

Кінцева оцінка обчислюється як зважена сума чотирьох нормалізованих компонентів та округлюється до шести знаків після коми. Для кожного апартаменту зберігається об'єкт ScoreBreakdown із внеском кожного критерію: PriceContribution, RatingContribution, DistanceContribution та CompletenessContribution. Ця деталізація передається клієнту й відображається у вигляді підказки при наведенні на картку апартаменту на сторінці результатів. На рисунку 4.7 зображено сторінку результатів пошуку.

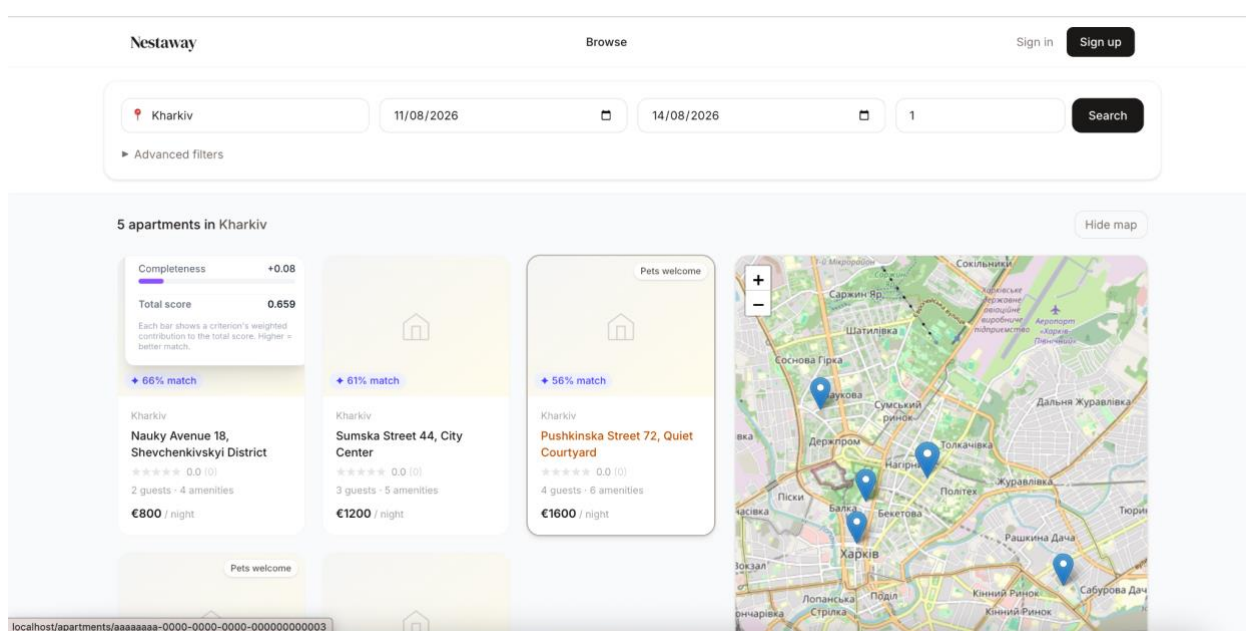


Рисунок 4.7 – сторінка результатів пошуку з розгорнутою підказкою ScoreBreakdown

4.6 Семантичний пошук: інтеграція OpenAI та pgvector

Семантичний пошук реалізований у класі SemanticSearchHandler і охоплює три послідовні фази. На першій фазі текст природномовного запиту передається до OpenAI API для генерації embedding-вектора через сервіс OpenAiEmbeddingService. Клієнт OpenAI SDK є потокобезпечним і реєструється як синглтон у контейнері залежностей. Модель та API-ключ

налаштовуються через секцію OpenAI конфігурації, що дозволяє змінити модель без перекомпіляції. Поточна модель – text-embedding-3-small – генерує вектори розмірністю 1536 вимірів.

На другій фазі виконується стандартний конвеєр фільтрів – ідентичний до звичайного пошуку – і отримується список відфільтрованих кандидатів. На третій фазі виконується запит до pgvector із обчисленням косинусної відстані між query-вектором та embedding-ами апартаментів, реалізація якого представлена у лістингу 4.2.

Лістинг 4.2 – Запит до pgvector із обчисленням косинусної відстані

```
var semRaw = await db.Apartments
    .Where(a => candidateIds.Contains(a.Id)
        && a.DescriptionEmbedding != null)
    .Select(a => new { a.Id,
        Distance =
a.DescriptionEmbedding!.CosineDistance(queryVector) })
    .OrderBy(x => x.Distance)
    .Take(20)
    .ToListAsync(ct);
```

Запит обмежує вибірку лише відфільтрованими кандидатами і повертає двадцять найближчих за семантичною схожістю об'єктів. Фінальна оцінка кожного результату обчислюється як лінійна комбінація: $FinalScore = 0.4 \times semantic_similarity + 0.6 \times ranking_score$, де $semantic_similarity = 1 - cosine_distance$. Такий розподіл ваг надає пріоритет якісним структурним характеристикам об'єкта, але дозволяє семантичному складнику суттєво впливати на ранжування при значних відмінностях у релевантності тексту.

Сервіс реалізує стратегію graceful degradation: якщо OpenAI API недоступне або повертає помилку, виключення перехоплюється через блок try-catch, а пошук продовжується без семантичного компонента,

повертаючи результати, ранжовані лише за структурними критеріями. У відповіді включається поле `Warning` з описовим повідомленням, що інформує клієнтську частину про тимчасову недоступність AI-компонента і дозволяє відобразити відповідне сповіщення користувачу.

`Embedding`-и апартаментів генеруються асинхронно як фонові задачі при публікації або оновленні оголошення. `GenerateEmbeddingJob`, запланована через `Hangfire`, отримує текст опису апартаменту, генерує вектор і зберігає його у поле `DescriptionEmbedding`. Оскільки задача є асинхронною, публікація оголошення не блокується доступністю `OpenAI API`.

4.7 Підсистема бронювання та обробка конфліктів

`CreateBookingCommand` реалізує надійне запобігання конфлікту даних через комбінацію песимістичного блокування на рівні бази даних та перевірки перекриття в рамках транзакції. При отриманні команди бронювання відкривається явна транзакція `PostgreSQL`, після чого апартамент завантажується з використанням оператора `SELECT ... FOR UPDATE`. Реалізація цього запиту представлена в лістингу 4.3.

Лістинг 4.3 – Реалізація `SELECT ... FOR UPDATE` при створенні бронювання

```
var apartment = await db.Apartments
    .FromSqlRaw(
        @"SELECT * FROM ""Apartments""
        WHERE ""Id"" = {0} FOR UPDATE",
        req.ApartmentId)
    .AsNoTracking()
    .FirstOrDefaultAsync(ct)
    ?? throw new
        NotFoundException(nameof(Apartment), req.ApartmentId);
```

Це блокування рядка гарантує, що жоден паралельний запит на бронювання того самого апартаменту не зможе одночасно пройти перевірку доступності, поки поточна транзакція не завершиться. В середині транзакції виконуються послідовні перевірки: активність апартаменту, відповідність мінімальної тривалості перебування, відповідність кількості гостей місткості, відсутність перекриваючих активних бронювань та відсутність явно заблокованих дат від орендодавця. При провалі будь-якої перевірки повертається `ConflictException`, а транзакція відкочується.

Обраний підхід має характерний недолік песимістичного блокування: у сценаріях із високою конкурентністю. Коли кілька користувачів одночасно намагаються забронювати один і той самий апартамент транзакції вишиковуються в чергу очікування. Тривалість утримання блокування визначається часом виконання всієї транзакції, включаючи мережеві затримки та перевірки бізнес-логіки. Це обмежує горизонтальне масштабування компонента бронювань і може призводити до деградації продуктивності під пікове навантаження.

Сторінка деталей апартаменту з реалізованою можливістю бронювання зображено на рисунку 4.8.

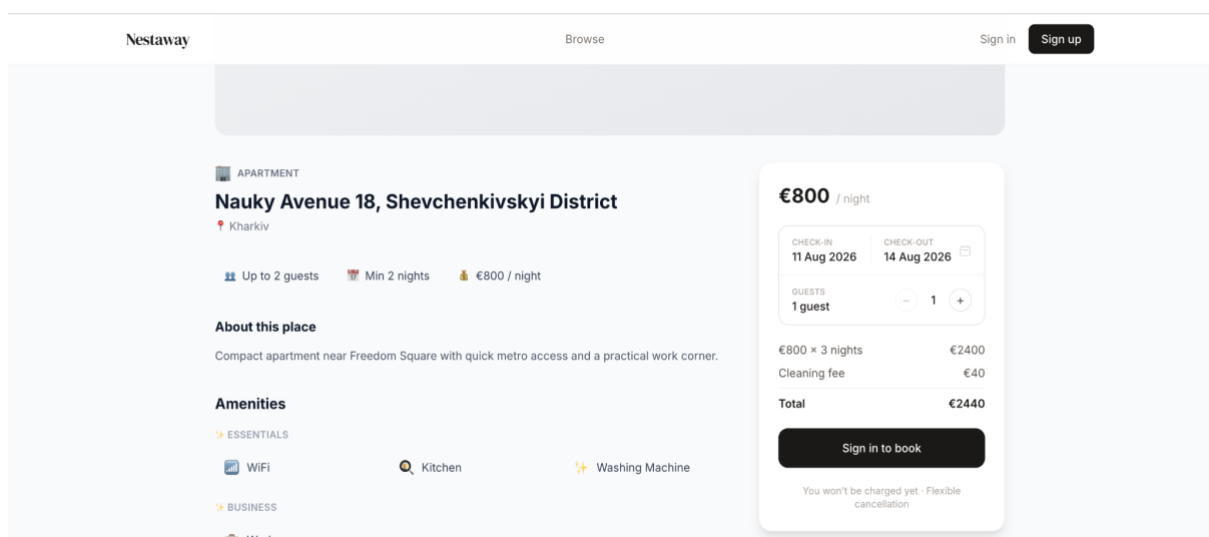


Рисунок 4.8 – сторінка деталей апартаменту

RefundCalculator обчислює суму повернення коштів виходячи з двох параметрів: політики скасування апартаменту та кількості днів до заїзду. Три доступні політики визначають різні порогові значення: Flexible – повне повернення за ≥ 1 день до заїзду, нульове при скасуванні в день заїзду; Moderate – повне повернення за ≥ 5 днів, 50% при 1–4 днях, нульове в день заїзду; Strict – повне повернення за ≥ 14 днів, 50% при 7–13 днях, нульове менш ніж за 7 днів. Реалізація використовує конструкцію switch expression, що представлений в лістингу 4.4.

Лістинг 4.4 – switch case, відповідальний за обчислення суми повернення коштів

```
return (policy, daysUntil) switch {
    (CancellationPolicy.Flexible, >= 1) =>
totalPrice,
    (CancellationPolicy.Moderate, >= 5) =>
totalPrice,
    (CancellationPolicy.Moderate, >= 1) =>
Round(totalPrice * 0.50m),
    (CancellationPolicy.Strict, >= 14) =>
totalPrice,
    (CancellationPolicy.Strict, >= 7) =>
Round(totalPrice * 0.50m),
    _ => 0m,
};
```

При скасуванні бронювання, крім безпосередніх змін у базі даних, виконується інвалідація кешу пошуку в Redis. Оскільки скасоване бронювання звільняє дати апартаменту, кешовані результати пошуку стають застарілими. Реалізація перебирає всі ключі з патерном search:v1:* та видаляє їх. Помилка при інвалідації кешу лише логується і не призводить до відкату операції – це свідоме архітектурне рішення, що надає пріоритет узгодженості основних даних над актуальністю кешу.

4.8 Фонові задачі та планувальник Hangfire

Система використовує Hangfire як планувальник фонових задач із бекендом на базі PostgreSQL. Це означає, що черги задач та їхній стан зберігаються в тій самій базі даних, що й основні дані застосунку, без необхідності підтримувати окремий сервіс-брокер повідомлень. Hangfire надає вбудований веб-інтерфейс адміністрування, доступний за адресою /hangfire.

У системі визначено чотири типи задач. `GenerateEmbeddingJob` запускається по завданню при публікації або оновленні оголошення і відповідає за генерацію та збереження векторного представлення опису в таблиці `Apartments`. `SendBookingConfirmationEmailJob` також є задачею по завданню і надсилає лист при підтвердженні бронювання. `RecalculateApartmentRatingsJob` є рекурентною задачею з розкладом `Cron.Hourly()`, що перераховує середній рейтинг для всіх апартаментів на основі активних відгуків. `WarmSearchCacheJob` запускається кожні шість годин і попередньо наповнює кеш Redis результатами типових пошукових запитів для популярних міст.

Паралельно з Hangfire реалізований запасний механізм обробки `embedding`-запитів у пам'яті процесу. `EmbeddingQueue` є каналною чергою (`System.Threading.Channels.Channel<T>`), а `EmbeddingWorker` є фоновією службою ASP.NET Core (`IHostedService`), що безперервно обслуговує цю чергу. У продуктивному середовищі Hangfire перехоплює всі задачі, але наявність резервного механізму гарантує, що при проблемах із Hangfire-бекендом `embedding`-и все одно будуть генеруватися.

4.9 Клієнтський застосунок

Клієнтська частина системи реалізована на React 18 з TypeScript. Управління глобальним станом здійснюється за допомогою Redux Toolkit, а

серверний стан і кешування запитів – через RTK Query. Маршрутизація реалізована за допомогою React Router v6 із вкладеними маршрутами та компонентом-оберткою ProtectedRoute, що перевіряє наявність токена автентифікації та відповідність ролі перед рендерингом захищеної сторінки. При відсутності токена або невідповідності ролі відбувається автоматичне перенаправлення на сторінку входу.

Домашня сторінка (HomePage) реалізує 2 режими пошуку. Перший режим – «AI search» – відображає компонент SemanticSearchBar, де користувач вводить довільний запит і отримує результати, ранжовані за семантичною релевантністю в поєднанні зі структурними критеріями. Другий режим – «Filter search» – відображає форму зі стандартними полями, що ініціює звичайний структурований пошук із ранжуванням. Обидва режими повертають результати у вигляді уніфікованого списку карток апартаментів, що рендериться компонентом ApartmentList. Кожна картка відображає ключові характеристики об'єкта – назву, ціну, рейтинг, фотографію та основні зручності – і є інтерактивним елементом, що веде на сторінку детального перегляду. На рисунку 4.9 зображено головну сторінку.

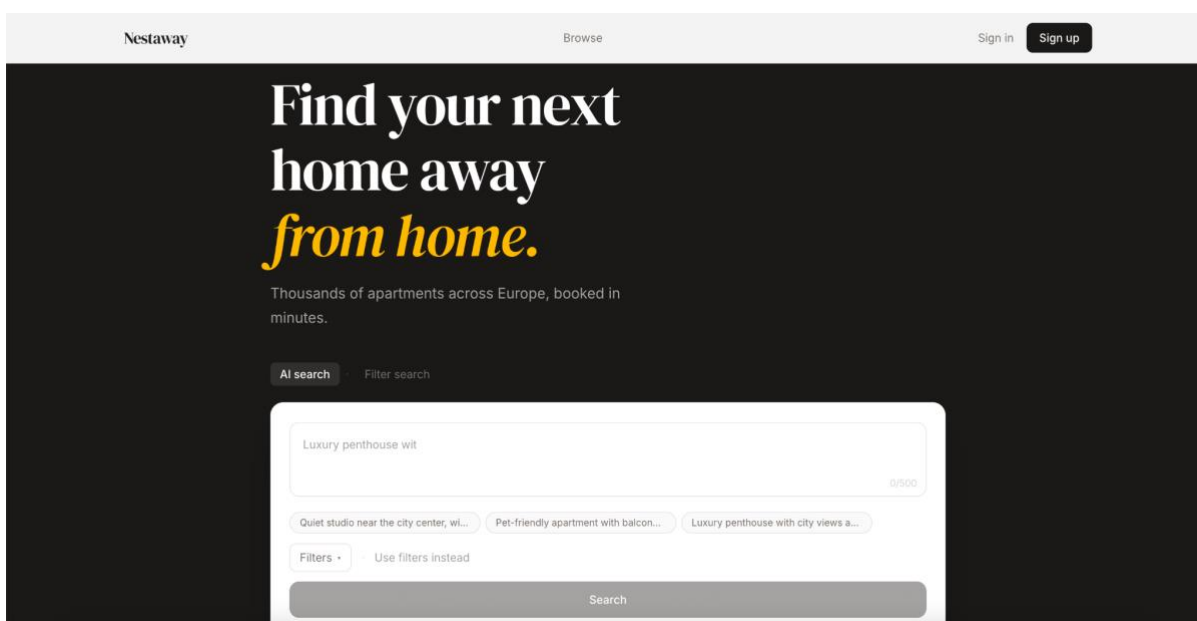


Рисунок 4.9 – головна сторінка

Сторінка результатів пошуку (SearchResultsPage) відображає картки апартamentів (ApartmentCard), кожна з яких у ранжованому режимі містить компонент ScoreBreakdownTooltip – підказку з деталізацією оцінки при наведенні. Сторінка результатів пошуку зображена на рисунку 4.10. Поруч зі списком розташована інтерактивна карта SearchMap на базі Leaflet.js із маркерами апартamentів, клік на якому виконує плавне прокручування до відповідної картки у списку. Компонент CityAutocomplete реалізує автодоповнення назв міст через RTK Query з debounce-затримкою для уникнення надмірних запитів.

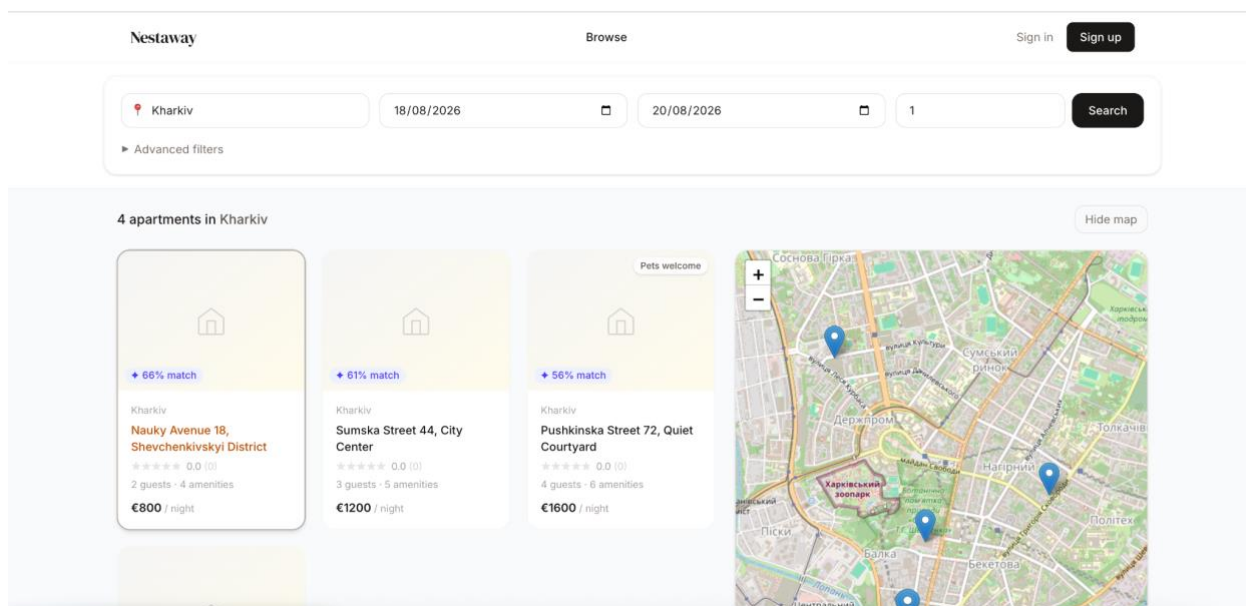


Рисунок 4.10 – сторінка результатів пошуку

Сторінка деталей апартamentу (ApartmentDetailPage) складається з кількох компонентів, кожен із яких відповідає за окрему частину інтерфейсу. Компонент PhotoCarousel реалізує карусель фотографій із можливістю перемикання між слайдами. AmenitiesGrid відображає перелік доступних зручностей у вигляді сітки з іконками та підписами. AvailabilityCalendar показує календар із позначенням доступних і заброньованих дат. ReviewsList виводить список відгуків попередніх гостей

із підтримкою пагінації. HostCard містить основну інформацію про орендодавця та кнопку для зв'язку з ним. BookingWidget є формою бронювання, що дозволяє обрати дати, вказати кількість гостей і розрахувати підсумкову вартість. На рисунку 4.11 зображена сторінка деталей апартаменту.

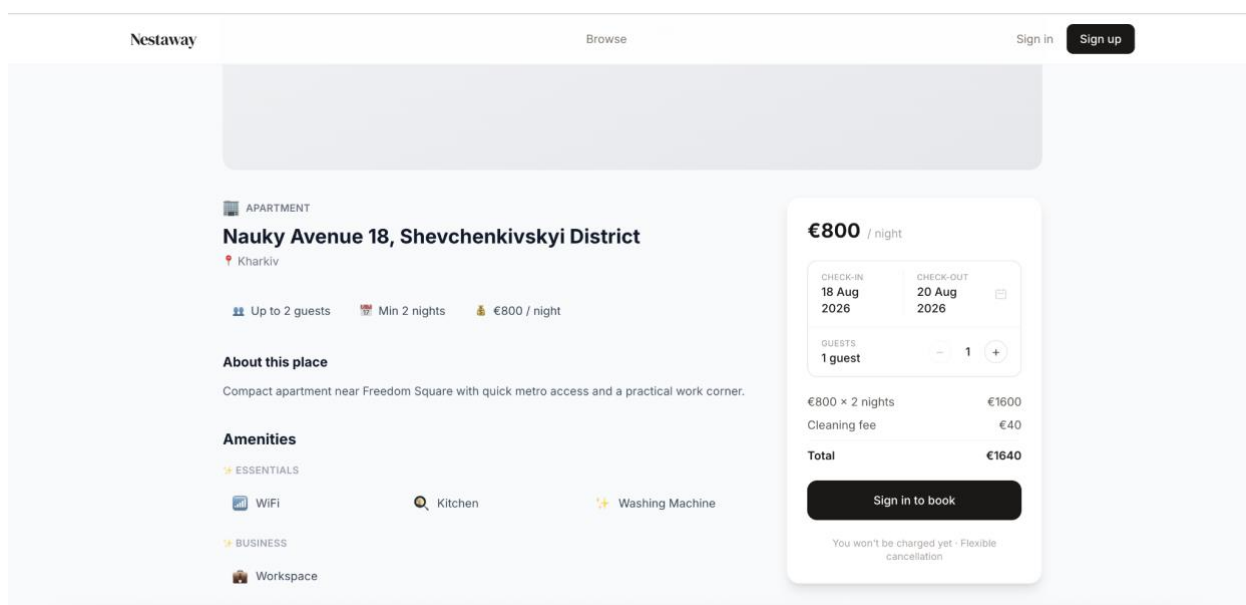


Рисунок 4.11 – сторінка деталей апартаменту (RankingPanel та BookingWidget)

Кабінет орендодавця (LandlordDashboard) включає три функціональні блоки: перелік власних оголошень із можливістю редагування та зміни статусу (MyListings), форму публікації нового об'єкта (CreateListing) та перелік вхідних запитів на бронювання для підтвердження або відхилення (IncomingBookings). Компонент AvailabilityManager дозволяє орендодавцю блокувати конкретні діапазони дат для обраного об'єкта. Панель адміністратора (AdminPanel) надає зведений огляд усіх об'єктів, бронювань та користувачів із можливістю деактивації облікового запису або видалення оголошення. Всі мутаційні запити до API реалізовані через RTK Query мутації з автоматичним інвалідуванням кешованих тегів після

успішного виконання. Сторінку, що реалізує кабінет орендодавця зображено на рисунку 4.12.

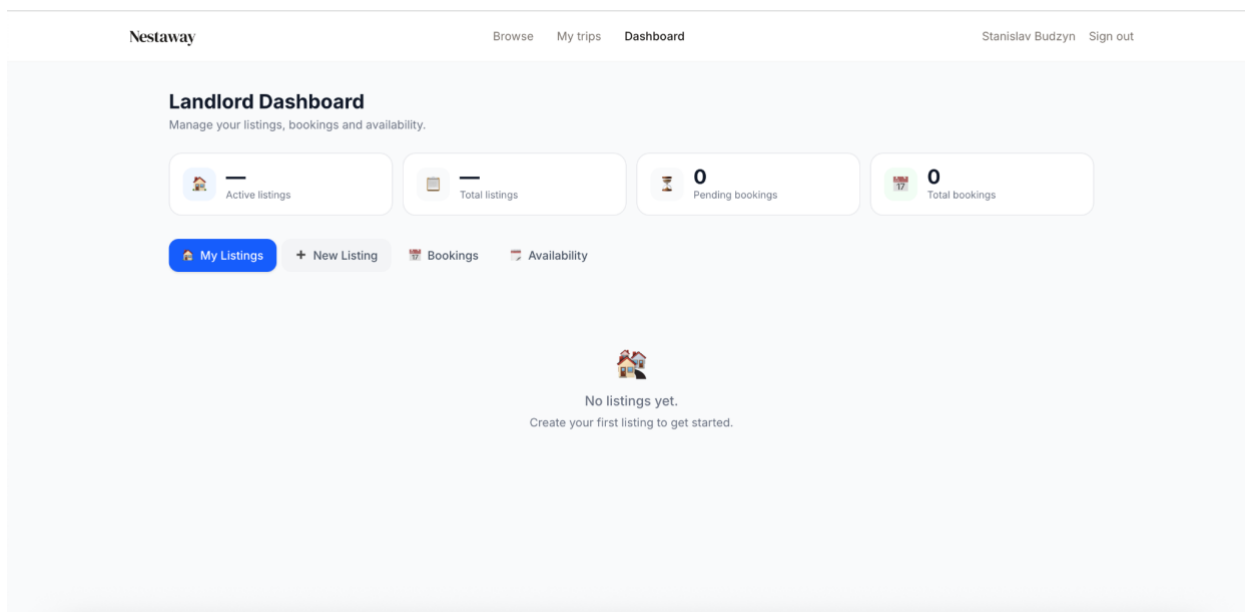


Рисунок 4.12 – LandlordDashboard

4.10 Контейнеризація та розгортання системи

Систему розгорнуто у вигляді набору Docker-контейнерів, описаних у файлі `docker-compose.yml`, що включає п'ять сервісів: `postgres`, `redis`, `backend`, `frontend` та `nginx`. Усі чутливі параметри конфігурації – паролі бази даних, секрет JWT, API-ключ OpenAI – передаються виключно через змінні оточення з файлу `.env` і не містяться у репозиторії. Файл `.env.example` зберігається в репозиторії і містить повний перелік необхідних змінних без значень.

Для PostgreSQL використовується власний Docker-образ, що будується з кастомного `Dockerfile` у директорії `docker/postgres`. Цей образ базується на офіційному `postgres:16` з попередньо встановленими розширеннями `PostGIS` та `pgvector`. Ініціалізаційний SQL-скрипт `init.sql` монтується як файл ініціалізації та виконується при першому запуску

контейнера. Перевірка готовності сервісу реалізована через healthcheck з командою pg_isready: контейнер backend не запуститься раніше, ніж база даних буде готова приймати з'єднання. Аналогічний healthcheck визначено для Redis із командою redis-cli ping.

Nginx-сервер виступає зворотним проксі: запити з префіксами /api/ та /hangfire/ перенаправляються до контейнера backend, решта – до контейнера frontend. Конфігурація Nginx також обробляє TLS-термінацію через сертифікати, що монтуються з директорії nginx/certs, що дозволяє обом внутрішнім сервісам використовувати незашифрований HTTP у межах внутрішньої Docker-мережі apartment_net із драйвером bridge.

Такий підхід до розгортання забезпечує повну відтворюваність середовища: будь-яка машина з встановленим Docker та Docker Compose може підняти повністю функціональну версію системи єдиною командою docker-compose up --build, що суттєво спрощує як локальну розробку, так і розгортання на виробничих серверах.

4.11 Висновки по розділу

Було описано конкретну програмну реалізацію системи бронювання апартаментів. Показано, що архітектурне рішення на основі Clean Architecture та CQRS з бібліотекою MediatR дозволяє чітко розмежувати відповідальності між шарами і зробити кожен компонент незалежно тестованим. Реалізація конвеєра пошуку на основі ланцюга фільтрів ISearchFilter забезпечує компонування довільної кількості критеріїв в єдиний SQL-запит без деградації продуктивності завдяки відкладеній матеріалізації LINQ-виразів.

Алгоритм ранжування у класі RankingService реалізує зважену багатокритеріальну оцінку з байєсівською корекцією рейтингів та підтримкою персоналізованих вагових коефіцієнтів для автентифікованих користувачів. Семантичний пошук у SemanticSearchHandler інтегрує

OpenAI Embeddings API з pgvector і реалізує стратегію graceful degradation при недоступності зовнішнього сервісу. Підсистема бронювання використовує SELECT FOR UPDATE для серіалізації конкурентних запитів та RefundCalculator для обчислення повернень відповідно до обраної політики скасування.

Фонова обробка за допомогою Hangfire забезпечує асинхронну генерацію embedding-ів, рекурентний перерахунок рейтингів та прогрівання кешу без блокування основного потоку обробки запитів. Клієнтський React-застосунок реалізує подвійний режим пошуку, рольову маршрутизацію та прозоре відображення оцінок ранжування для кінцевого користувача. Система розгорнута у вигляді п'яти Docker-контейнерів із Nginx як зворотним проксі, що забезпечує відтворюваність середовища та ізоляцію компонентів.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне завдання проєктування та реалізації інформаційної системи бронювання апартаментів, що поєднує традиційний структурований пошук із методами штучного інтелекту для підвищення релевантності результатів та якості взаємодії з користувачем.

У першому розділі виконано аналіз предметної галузі короткострокової оренди житла та встановлено ключові проблеми, що обмежують ефективність наявних платформ: надлишок пропозицій без змістовного ранжування, неоднорідність даних оголошень, динамічне ціноутворення та відсутність персоналізації під конкретний контекст запиту. Проведено огляд функціонального ядра сучасних систем бронювання та їхньої еволюції – від пасивних каталогів до активних посередників між запитом і пропозицією. Визначено, що задовільного рішення проблеми ранжування при великій кількості об'єктів без урахування семантики запиту та індивідуального контексту користувача бракує у більшості розглянутих систем.

У другому розділі проаналізовано та обґрунтовано технологічний стек системи. На основі порівняльного аналізу обрано ASP.NET Core 8 для серверної частини як платформу з розвиненою підтримкою асинхронного програмування та інтеграцій із зовнішніми сервісами, PostgreSQL як основну реляційну СУБД із розширеннями PostGIS та pgvector для геопросторових запитів та векторного пошуку відповідно, Redis для кешування та зберігання refresh-токенів, React з TypeScript для клієнтського застосунку. Розглянуто архітектурні підходи, зокрема чисту архітектуру та CQRS, обґрунтовано їх вибір для забезпечення розширюваності та тестованості системи. Проаналізовано існуючі підходи до семантичного пошуку та ранжування в інформаційних системах.

У третьому розділі спроектовано архітектуру AI-компонентів системи. Розроблено багатокритеріальний зважений алгоритм ранжування,

що враховує ціну, байєсівський рейтинг, географічну відстань та повноту профілю оголошення. Для алгоритму обґрунтовано базові вагові коефіцієнти та механізм їх персоналізації на основі збережених уподобань автентифікованого користувача. Спроектовано архітектуру семантичного пошуку на основі векторних представлень OpenAI та косинусної подібності `pgvector`, де кінцева оцінка є зваженою комбінацією семантичного та структурного компонентів. Детально описано компонент автоматичної оцінки якості текстових описів оголошень, реалізований через звернення до OpenAI API, та систему рекомендацій схожих об'єктів на основі попередньо розрахованих векторних сусідств. Для кожного AI-компонента визначено стратегію *graceful degradation*, що забезпечує коректну роботу системи при недоступності зовнішніх сервісів.

Проведено оцінку адекватності AI-модуля, що охоплює всі чотири компоненти системи. Для кожного компонента визначено відповідність обраного методу природі задачі, що вирішується, та виявлено як переваги, так і обмеження прийнятих проєктних рішень. Встановлено, що застосування векторних представлень є адекватним для подолання *vocabulary mismatch* у предметній галузі, де описи об'єктів є достатньо змістовними для формування інформативного вектора. Підхід до персоналізації на основі корекцій вагових коефіцієнтів визнано адекватним для поточного масштабу системи як такий, що не потребує критичної маси користувачів на відміну від колаборативної фільтрації. Застосування мовної моделі для оцінки якості опису визнано єдиним практично прийнятним підходом для задачі, що не має детерміністичного аналога. Виявлені обмеження (евристичність вікна агрегації поведінкових даних, стохастична варіативність оцінок мовної моделі та статичність кешованих рекомендацій) є прийнятними компромісами для поточного етапу розробки та усуненими в наступних ітераціях без зміни загальної архітектури системи.

У четвертому розділі реалізовано систему у повному обсязі. Серверна частина побудована за принципами чистої архітектури з використанням

паттерну CQRS та бібліотеки MediatR, що забезпечує чіткий поділ відповідальностей між шарами. Конвеєр пошуку реалізовано як ланцюг незалежних фільтрів `ISearchFilter`, що компонуються в єдиний SQL-запит завдяки відкладеній матеріалізації LINQ-виразів. Підсистема бронювання використовує `SELECT FOR UPDATE` для серіалізації конкурентних запитів та тришарову логіку повернення коштів залежно від обраної політики скасування. Фонову обробку – генерацію `embedding`-ів, перерахунок рейтингів та прогрівання кешу – реалізовано через планувальник Hangfire з бекендом на PostgreSQL. Клієнтський застосунок реалізовано на React 18 з TypeScript та Redux Toolkit, з підтримкою подвійного режиму пошуку (семантичний та структурний), рольової маршрутизації та прозорого відображення складових оцінки ранжування для кожного результату. Систему контейнеризовано засобами Docker Compose у складі п'яти сервісів із Nginx як зворотним проксі.

Реалізована підсистема бронювання має відоме архітектурне обмеження, пов'язане із застосуванням песимістичного блокування. Оператор `SELECT FOR UPDATE` серіалізує всі конкурентні транзакції на рівні рядка бази даних. У сценарії, коли кілька користувачів одночасно намагаються забронювати один апартамент, запити вишиковуються в чергу очікування, а тривалість утримання блокування визначається часом виконання всієї транзакції, включаючи мережеві затримки та перевірки бізнес-логіки. Це обмежує горизонтальне масштабування компонента бронювань і може призводити до деградації продуктивності під пікове навантаження. Для усунення цього обмеження в майбутніх ітераціях розробки доцільно розглянути перехід до оптимістичного керування конкурентністю на основі поля версії, яке дозволяє паралельне виконання транзакцій і відхиляє лише ті, що фактично конфліктують.

Практична цінність роботи полягає в тому, що реалізована система демонструє ефективну інтеграцію AI-методів – векторного семантичного пошуку та персоналізованого багатокритеріального ранжування – у

повноцінний web-застосунок із продуктивним стеком реального рівня. Усі AI-компоненти спроектовані за принципом graceful degradation і не знижують загальну надійність системи при відмові зовнішніх сервісів. Архітектурні рішення, описані в роботі, можуть бути адаптовані для широкого кола задач пошуку та ранжування в інших предметних областях, де поєднання структурованих фільтрів та семантичного розуміння запиту дає якісно кращий результат, ніж кожен підхід окремо.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Zervas G., Proserpio D., Byers J. W. The rise of the sharing economy: estimating the impact of airbnb on the hotel industry. *Journal of marketing research*. 2017. Vol. 54, no. 5. P. 687–705. URL: <https://doi.org/10.1509/jmr.15.0204> (date of access: 18.04.2026).
2. Adomavicius G., Tuzhilin A. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*. 2005. Vol. 17, no. 6. P. 734–749. URL: <https://doi.org/10.1109/tkde.2005.99> (date of access: 18.04.2026).
3. Recommender systems handbook / ed. by F. Ricci, L. Rokach, B. Shapira. Boston, MA : Springer US, 2015. URL: <https://doi.org/10.1007/978-1-4899-7637-6> (date of access: 18.04.2026).
4. Hwang C.-L., Yoon K. Methods for multiple attribute decision making. *Multiple attribute decision making*. Berlin, Heidelberg, 1981. P. 58–191. URL: https://doi.org/10.1007/978-3-642-48318-9_3 (date of access: 02.05.2026).
5. Koren Y., Bell R., Volinsky C. Matrix factorization techniques for recommender systems. *Computer*. 2009. Vol. 42, no. 8. P. 30–37. URL: <https://doi.org/10.1109/mc.2009.263> (date of access: 04.05.2026).
6. Linden G., Smith B., York J. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE internet computing*. 2003. Vol. 7, no. 1. P. 76–80. URL: <https://doi.org/10.1109/mic.2003.1167344> (date of access: 04.05.2026).
7. Farronato C., Fradkin A. The welfare effects of peer entry in the accommodation market: the case of airbnb. Cambridge, MA : National bureau of economic research, 2018. URL: <https://doi.org/10.3386/w24361> (date of access: 10.05.2026).

8. Why tourists choose airbnb: a motivation-based segmentation study / D. Guttentag et al. *Journal of travel research*. 2017. Vol. 57, no. 3. P. 342–359. URL: <https://doi.org/10.1177/0047287517696980> (date of access: 10.05.2026).
9. Jakob N. Usability engineering. Boston : Academic Press, 1993. 358 p.
10. Salton G., Buckley C. Term-weighting approaches in automatic text retrieval. *Information processing & management*. 1988. Vol. 24, no. 5. P. 513–523. URL: [https://doi.org/10.1016/0306-4573\(88\)90021-0](https://doi.org/10.1016/0306-4573(88)90021-0) (date of access: 11.05.2026).
11. Mikolov T., Chen K., Corrado G., Dean J. Efficient estimation of word representations in vector space, 2013. URL: <https://doi.org/10.48550/arXiv.1301.3781> (date of access: 13.05.2026)
12. Vector embeddings | OpenAI API. *OpenAI Developers* URL: <https://platform.openai.com/docs/guides/embeddings> (date of access: 14.05.2026).
13. Malkov Y. A., Yashunin D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*. 2020. Vol. 42, no. 4. P. 824–836. URL: <https://doi.org/10.1109/tpami.2018.2889473> (date of access: 20.05.2026).
14. Methods and metrics for cold-start recommendations / A. I. Schein et al. *The 25th annual international ACM SIGIR conference*, Tampere, Finland, 11–15 August 2002. New York, New York, USA, 2002. URL: <https://doi.org/10.1145/564376.564421> (date of access: 20.05.2026).
15. Item-based collaborative filtering recommendation algorithms / B. Sarwar et al. *The tenth international conference*, Hong Kong, Hong Kong, 1–5 May 2001. New York, New York, USA, 2001. URL: <https://doi.org/10.1145/371920.372071> (date of access: 20.05.2026).
16. Clean architecture: a craftsman's guide to software structure and design. Prentice Hall, 2017. 432 p. (date of access: 21.05.2026)

17. GitHub – pgvector/pgvector: open-source vector similarity search for Postgres. *GitHub*. URL: <https://github.com/pgvector/pgvector> (date of access: 01.06.2026).
18. Guttentag D. Airbnb: disruptive innovation and the rise of an informal tourism accommodation sector. *Current issues in tourism*. 2013. Vol. 18, no. 12. P. 1192–1217. URL: <https://doi.org/10.1080/13683500.2013.827159> (date of access: 01.06.2026).
19. Buhalis D., Law R. Progress in information technology and tourism management: 20 years on and 10 years after the Internet–The state of eTourism research. *Tourism management*. 2008. Vol. 29, no. 4. P. 609–623. URL: <https://doi.org/10.1016/j.tourman.2008.01.005> (date of access: 19.05.2026).
20. Decision support systems: concepts and resources for managers. Quorum Books, 2002. 225 p.
21. Tussyadiah I. P. Toward a theoretical foundation for experience design in tourism. *Journal of travel research*. 2013. Vol. 53, no. 5. P. 543–564. URL: <https://doi.org/10.1177/0047287513513172> (date of access: 19.05.2026).
22. BERT: pre-training of deep bidirectional transformers for language understanding / J. Devlin et al. *Proceedings of the 2019 conference of the north, Minneapolis, Minnesota. Stroudsburg, PA, USA, 2019*. URL: <https://doi.org/10.18653/v1/n19-1423> (date of access: 21.05.2026).
23. Johnson J., Douze M., Jegou H. Billion-scale similarity search with GPUs. *IEEE transactions on big data*. 2019. P. 1. URL: <https://doi.org/10.1109/tbdata.2019.2921572> (date of access: 22.05.2026).
24. Reimers N., Gurevych I. Sentence-BERT: sentence embeddings using siamese bert-networks. *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP), Hong Kong, China. Stroudsburg, PA, USA, 2019*. URL: <https://doi.org/10.18653/v1/d19-1410> (date of access: 22.05.2026).