ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Вебзастосунок інформаційно-пошукової системи для оренди житла»

Ступінь вищої освіти – бакалавр
Спеціальність – 122 «Комп'ютерні науки»
Освітня програма «Комп'ютерні науки»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: здобувач 4 курсу
групи КН-21
Валентин БОНДАРЕНКО
Керівник: ст. викладач кафедри комп'ютерних
наук
Олег ЛУКУТІН

Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.
Здобувач: _____
(підпис)

м. Київ – 2025 рік

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри
комп'ютерних наук
_____Сергій МІЧКІВСЬКИЙ
«____»____20___р

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бондаренко Валентин Олегович

Тема роботи	Вебзастосунок інформаційно-пошукової системи для оренди житла
Номер та дата наказу про затвердження теми	№121-7 від 24 грудня 2024 року
Коротка постановка завдання	Розробка веб-застосунку інформаційно-пошукової системи для оренди житла з рекомендаційною системою та певними критеріальними фільтрами
Посилання на джерела інформації (не більше п'яти найменувань, які рекомендує науковий керівник)	1. Федько М.Р. Дослідження методів побудови рекомендаційних систем для вирішення задач в електронній комерції, методи колаборативної фільтрації. // International Electronic Scientific Journal "Science Online", 2021. URL https://uk.legacy.reactjs.org/ 2. Шаповалов С.М., Мічківський С.М. Розробка рекомендаційної системи для підбору співрозмовника в соціальній мережі // XII Міжнародна науково-технічна конференція «Проблеми Інформатизації» (м. Київ, 13 грудня 2018 року). – Київ: Державний університет телекомунікацій, 2018. URL: https://dut.edu.ua/uploads/l_1701_65845854.pdf
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має передбачити теоретичне, системотехнічне або експериментальне дослідження складного спеціалізованого завдання або практичної проблеми в галузі комп'ютерних наук, яке характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів інформаційних технологій.

Дата видачі завдання 27 грудня 2024 р.

Керівник Олег ЛУКУТІН

Здобувач освітнього ступеня бакалавра Валентин БОНДАРЕНКО

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
	Підготовчий етап		
1	Вибір напряму дослідження	02.12.2024 р.	*виконано*
2	Формування теми та призначення керівника	16.12.2024 р.	*виконано*
3	Затвердження теми кваліфікаційної роботи	23.12.2024 р.	*виконано*
4	Затвердження завдання на кваліфікаційну роботу	27.12.2024 р.	*виконано*
	Основний етап		
5	Розробка концепції кваліфікаційної роботи	13.01.2025 р.	*виконано*
6	Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів	20.01.2025 р.	*виконано*
7	Затвердження розширеної постановки завдання. Підготовка та подання керівникові розділу 1 кваліфікаційної роботи	10.03.2025 р.	*виконано*
8	Проєктування. Підготовка та подання керівникові розділу 2 кваліфікаційної роботи	24.03.2025 р.	*виконано*
9	Підготовка доповіді для експертизи стану виконання кваліфікаційної роботи (проміжний контроль)	31.03-04.04.2025 р.	*виконано*
10	Реалізація. Підготовка та подання керівникові розділу 3 кваліфікаційної роботи	07.04.2025 р.	*виконано*
11	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи	14.04.2025 р.	*виконано*
12	Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівникові доопрацьованого варіанту кваліфікаційної роботи	21.04.2025 р.	*виконано*
	Завершальний етап		
13	Представлення рукопису для перевірки на плагіат	28.04-04.05.2025 р.	*виконано*
14	Підготовка презентації та доповіді на передзахист	05.05-11.05.2025 р.	*виконано*
15	Передзахист кваліфікаційної роботи	12.05-16.05.2025 р.	*виконано*
16	Доопрацювання роботи за результатами передзахисту	19.05-06.06.2025 р.	*виконано*
17	Експертиза роботи керівником та зовнішнім експертом	09.06-15.06.2025 р.	*виконано*
18	Доопрацювання доповіді та презентації для захисту	09.06-15.06.2025 р.	*виконано*
19	Захист кваліфікаційної роботи	16.06-22.06.2025 р.	*виконано*

Керівник Олег ЛУКУТІН

Здобувач освітнього ступеня бакалавра Валентин БОНДАРЕНКО

Бондаренко В.О. Вебзастосунок інформаційно-пошукової системи для оренди житла

Пояснювальна записка кваліфікаційної роботи за спеціальністю 122 – Комп'ютерні науки (освітня програма – Комп'ютерні науки) СО Бакалавр. – ВНЗ .Університет економіки та права .КРОК., Навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра комп'ютерних наук, Київ, 2025.

Описано Розробку веб-застосунку інформаційно-пошукової системи для оренди житла з рекомендаційною системою та певними критеріальними фільтрами.

Ключові слова: веб-сайт, гаджети, мобільні пристрої, рекомендації.

Малюнок. 9. Бібліограф. 11.

Bondarenko V.O. Web application of information and search system for rental housing

Explanatory note of qualification work in specialty 122 - Computer Science (educational programme - Computer Science), Bachelor's degree - University of Economics and Law "KROK", Educational and Research Institute of Information and Communication Technologies, Department of Computer Science, Kyiv, 2025.

The development of a web application of an information search system for renting housing with a recommendation system and certain criteria filters

Keywords: website, gadgets, mobile devices, recommendations.

Fig. 9. Bibliography. 11.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний світ свідчить про зростання попиту на оренду житла як серед місцевих мешканців, так і серед приїжджих туристів. Люди шукають зручні та доступні способи підбору нерухомості відповідно до своїх потреб та фінансових можливостей.

Одним із найефективніших інструментів для пошуку житла є веб-застосунки, які дозволяють користувачам швидко знайти відповідні пропозиції, використовуючи різні критерії фільтрації, такі як розташування, вартість, тип житла, додаткові зручності тощо.

Таким чином, розробка веб-застосунку інформаційно-пошукової системи для оренди житла з багатокритеріальними фільтрами та рекомендаційною системою є важливим та актуальним напрямком досліджень, що сприятиме спрощенню та покращенню процесу вибору житла для орендарів.

Мета дослідження. Метою проєкту є розробка програмного забезпечення веб-застосунку для оренди житла з використанням сучасних технологій та реалізацією функцій багатокритеріальних фільтрів і рекомендаційної системи підбору об'єктів нерухомості.

Для досягнення мети проєкту були виконані наступні завдання:

- розглянуто аналоги, які існують і користуються попитом станом на сьогоднішній день;
- визначено вимоги до програмного забезпечення системи;
- розглянуто методи та підходи до створення рекомендаційної системи та багатокритеріальних фільтрів;
- спроєктовано архітектуру програмного забезпечення веб-застосунку за допомогою UML-діаграм;
- спроєктовано базу даних;
- розроблено веб-інтерфейс, який дозволяє користувачам шукати, переглядати та порівнювати різні варіанти житла;

– створено систему фільтрації, що дозволяє клієнтам швидко знаходити житло відповідно до їхніх вимог та критеріїв, таких як місцезнаходження, ціна, площа, тип житла та інші параметри;

– розроблено алгоритм рекомендацій, який аналізує інформацію про користувачів та рекомендує об'єкти нерухомості, які найбільше відповідають їхнім запитам та попереднім виборам;

– розроблено програмне забезпечення системи, інтеграцію всіх компонентів системи та тестування їхньої працездатності.

Об’єктом дослідження є візуалізація та управління веб-контентом, методи/алгоритми фільтрації даних за заданими критеріями, методи та підходи до формування рекомендацій.

Предметом дослідження є вебзастосунок інформаційно-пошукової системи для оренди житла, який використовує багатокритеріальні фільтри та рекомендаційну систему для підбору оптимальних варіантів відповідно до потреб користувачів.

Методи дослідження. Дослідження проводилося з використанням різних **наукових** та дослідницьких методів, зокрема аналізу конкурентів для розуміння їхніх підходів та стратегій у сфері оренди житла. Крім того, у процесі розробки та вдосконалення веб-застосунку застосовувалися мови програмування та сучасні технології веб-розробки.

Практичне значення. Запровадження сучасних методів розробки підвищує надійність і швидкість обробки даних, а також покращує безпеку використання веб-технологій. Використання багатокритеріального пошуку пришвидшує процес підбору житла, а рекомендаційна система сприяє більш точному підбору варіантів відповідно до потреб користувачів.

Структура та обсяг пояснювальної записки. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та списку посилань (11 найменувань). Пояснювальна записка містить 9 рисунків. Загальний обсяг пояснювальної записки складає сторінок, основний зміст викладено на 46 сторінках.

РОЗДІЛ 1
ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ ВЕБЗАСТОСУНКУ ІНФОРМАЦІЙНО-ПОШУКОВОЇ СИСТЕМИ ДЛЯ ОРЕНДИ ЖИТЛА

1.1 Опис предметної області

Оренда житла - це форма цивільно-правових відносин, за якої одна сторона (власник житла) передає іншій стороні (орендареві) право тимчасового користування нерухомістю за певну плату і на визначений строк. У широкому сенсі, оренда житлових приміщень дозволяє особам, які не володіють власним житлом або не мають постійного місця проживання, забезпечити собі тимчасові житлові умови.

Необхідність оренди виникає в найрізноманітніших життєвих ситуаціях: під час переїзду на нове місце роботи, навчання в іншому місті, туристичних подорожей, тимчасового проживання під час ремонту або продажу власного житла, а також в умовах вимушеного переселення. Особливо актуальною ця потреба стала в Україні в останні роки через внутрішню міграцію та зміну житлових умов великої кількості громадян.

Оренда має низку переваг:

- вона забезпечує гнучкість у виборі місця проживання;
- дозволяє уникнути великих одноразових витрат на купівлю нерухомості;
- забезпечує можливість швидко змінити житло, якщо змінюються обставини.

Водночас пошук житла для оренди може супроводжуватися труднощами - такими як брак достовірної інформації, велика кількість оголошень без фільтрації, шахрайські схеми, дублікати, застарілі дані, неефективна комунікація з орендодавцями тощо. Тому ефективний інструмент пошуку**,** фільтрації та аналізу оголошень про оренду житла є не лише зручним, а й необхідним у сучасних умовах.

У цьому контексті особливе значення набувають інформаційно-пошукові системи, зокрема веб-застосунки, які забезпечують користувачеві централізований доступ до актуальних пропозицій оренди, можливість швидкого порівняння варіантів та прийняття обґрунтованого рішення. Створення такого застосунку дозволяє систематизувати та оптимізувати взаємодію між орендодавцями та орендарями, зменшити ризики, підвищити ефективність пошуку і рівень задоволеності користувача.

Інформаційно-пошукова система для оренди житла - це вебзастосунок, який надає користувачам можливість здійснювати пошук та бронювання житла через Інтернет. Вона дозволяє орендодавцям розміщувати оголошення про доступні об'єкти, а орендарям - швидко знаходити житло відповідно до своїх потреб та вподобань.

Основна функція веб-застосунку - багатокритеріальні фільтри, що дозволяють користувачам здійснювати пошук житла за різними параметрами, такими як розташування, ціна, площа, кількість кімнат, наявність меблів, термін оренди тощо. Це значно спрощує процес пошуку та допомагає знайти варіант, що найбільше відповідає вимогам користувача.

Створення онлайн-платформи для оренди житла надає власникам нерухомості ефективний інструмент для розміщення оголошень, управління бронюваннями та комунікації з потенційними орендарями. Користувачі можуть переглядати доступні варіанти, отримувати детальну інформацію про об’єкти та їх характеристики, а також швидко зв’язуватися з орендодавцями.

Такий підхід особливо корисний для приватних власників житла та невеликих агентств нерухомості, оскільки на великих платформах, наприклад, Airbnb або Booking, конкуренція дуже висока. Власна система для пошуку та оренди житла дозволяє залучати цільову аудиторію, спрощує процес комунікації між сторонами та забезпечує швидке знаходження оптимальних варіантів житла.

1.2 Огляд аналогів

На українському ринку існує кілька веб-платформ, які спеціалізуються на оренді житла та надають схожі функції:

OLX Нерухомість (рис. 1.1) - одна з найбільш популярних платформ в Україні для розміщення оголошень з оренди та продажу житла. Технічна реалізація ресурсу побудована на сучасних веб-технологіях, зокрема JavaScript-фреймворках, що забезпечує швидке завантаження контенту через API та адаптивність до мобільних пристроїв.

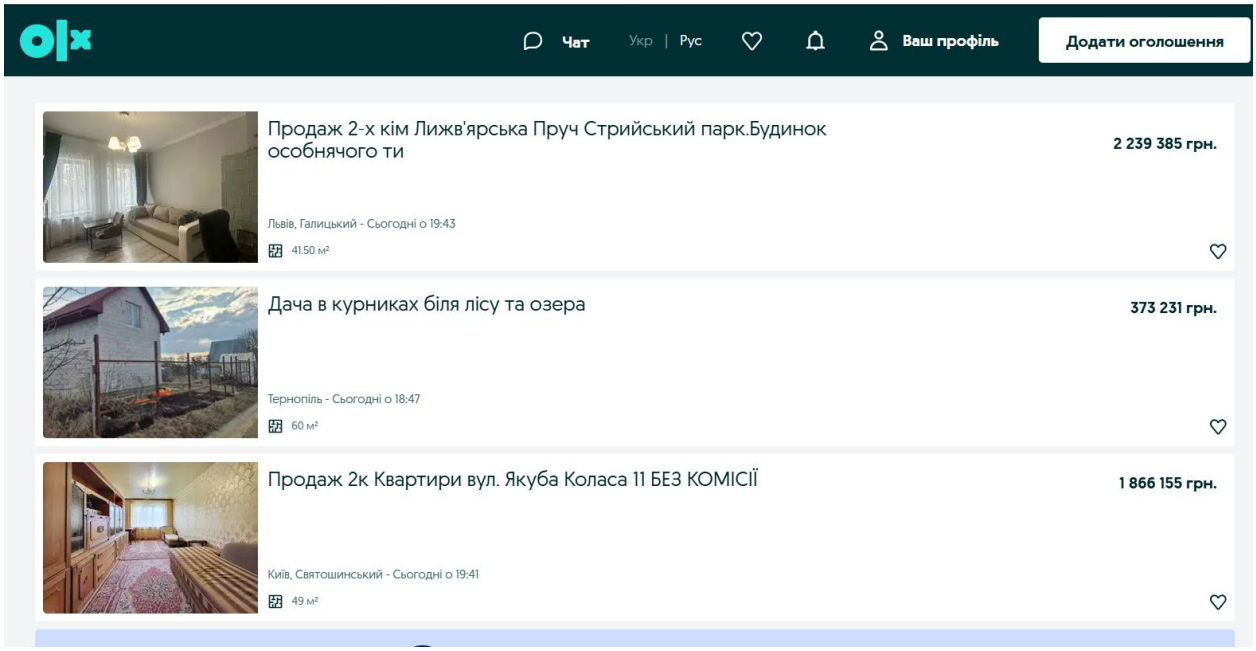

Рисунок 1.1 – сайт OLX Нерухомість

Джерело: [1]

Серед основних переваг ресурсу можна відзначити зрозумілий інтерфейс, базові фільтри пошуку, широку аудиторію користувачів та високу швидкодію. Разом із тим, платформа має низку обмежень: відсутність персоналізованих рекомендацій, слабку модерацію оголошень та перевантаження сторінок рекламними елементами. Пошукова система не

завжди враховує варіативність запитів, а відсутність адаптивної логіки у виборі пропозицій знижує ефективність користування.

З технічної точки зору, сайт може слугувати прикладом ефективної інтеграції REST API для динамічного завантаження контенту, базової фільтрації за параметрами та побудови масштабованої архітектури клієнт-сервер. Також платформа демонструє успішне застосування адаптивного дизайну для підтримки різних типів пристроїв, що є важливим аспектом у сучасній веброзробці.

Lun.ua (рис. 1.2) - агрегатор нерухомості, що надає детальну інформацію про квартири та будинки в оренду. Платформа реалізована з використанням сучасних клієнтських технологій, ймовірно на базі React або іншого JavaScript-фреймворка, що дозволяє ефективно відображати великі обсяги даних у реальному часі. Завдяки інтерактивним картам, гнучким фільтрам та додатковим інструментам (наприклад, аналітика цін, історія забудови), сайт пропонує користувачеві розширений функціонал для пошуку житла.

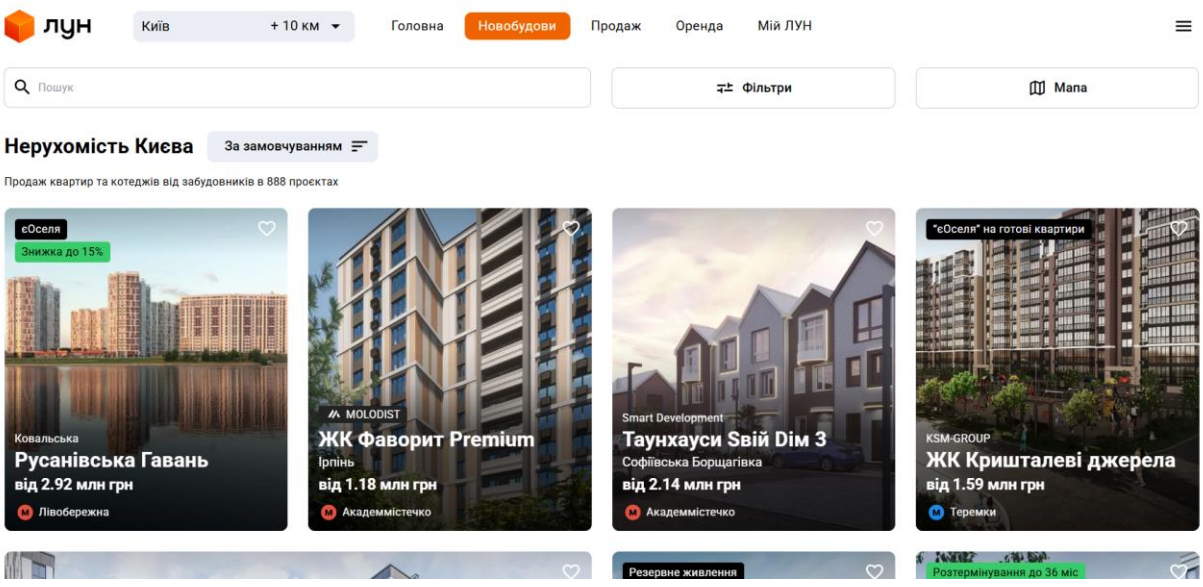

Рисунок 1.2 – сайт Lun.ua

Джерело: [2]

Серед технічних переваг Lun.ua варто виокремити високу якість візуалізації, інтеграцію з картографічними сервісами, динамічне завантаження контенту та структуровану подачу даних про об’єкти. Фільтрація дозволяє точно налаштовувати параметри пошуку за допомогою багатьох критеріїв - від ціни до типу будинку й наявності інфраструктури поруч. Окрему увагу розробники приділили зручності інтерфейсу: адаптивна верстка, інформативні карточки та візуально чистий дизайн позитивно впливають на користувацький досвід.

У технічному аспекті Lun.ua може бути корисним як приклад ефективної реалізації інтерактивної карти з геоприв’язкою об’єктів, складної багатокритеріальної фільтрації, а також використання зовнішніх API для збагачення даних. Крім того, платформа демонструє можливість поєднання стандартної пошукової моделі з аналітичними інструментами, що дозволяє користувачу оцінити ринкову ситуацію ще до перегляду конкретних об’єктів.

Ria.com Нерухомість (рис. 1.3) - спеціалізована платформа для пошуку житла, яка акцентує увагу на безпеці та достовірності оголошень. Технічна реалізація ресурсу передбачає обов’язкову модерацію контенту, перевірку фотографій та використання системи рейтингів орендодавців. Сайт також надає функцію перегляду історії об’єкта, що дозволяє користувачеві оцінити зміни в ціні, активність оголошення та наявність повторного розміщення.

З технічної точки зору, платформа демонструє впровадження важливих механізмів верифікації, зокрема ручної модерації та автоматичного аналізу зображень, що зменшує ймовірність фальшивих або неактуальних пропозицій. Інтерфейс сайту реалізований у класичному стилі, з адаптивною версткою та підтримкою основних функцій фільтрації - за розташуванням, типом житла, ціною та іншими параметрами. Швидкість завантаження сторінок стабільна завдяки використанню динамічного підвантаження контенту через API.

Платформа Ria.com може слугувати прикладом успішної інтеграції системи довіри між користувачем і сервісом через візуальні індикатори якості, рейтинги та перевірки. Водночас технічне рішення орієнтоване переважно на

якість, а не на обсяг, тому загальна кількість пропозицій може бути нижчою, ніж у конкурентів. Це створює баланс між безпекою та охопленням, що важливо враховувати при проєктуванні власного рішення.

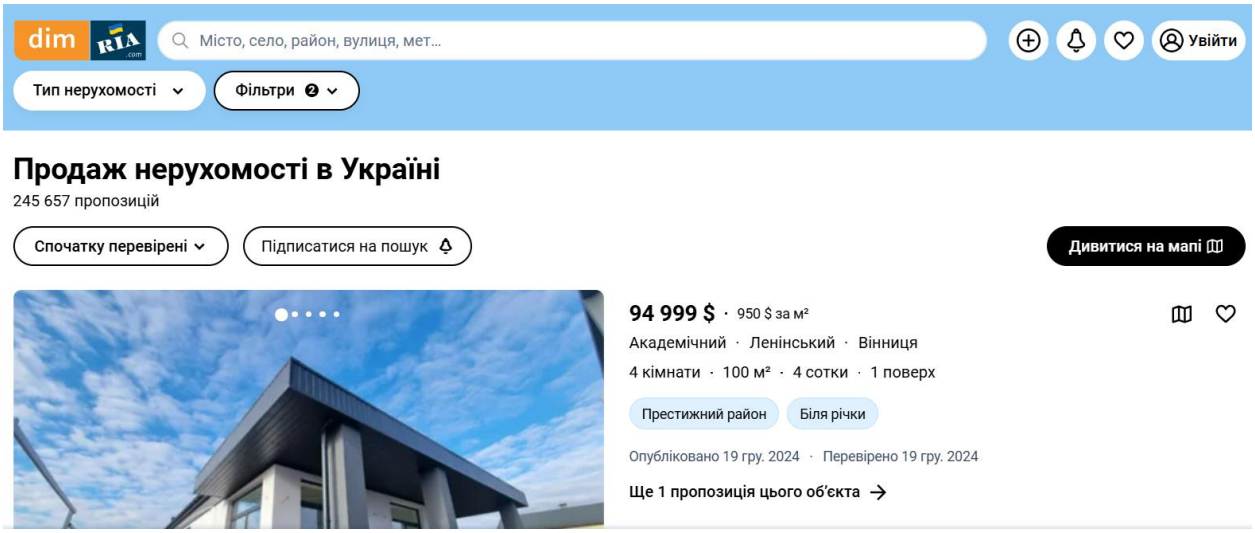

Рисунок 1.3 – сайт Ria.com Нерухомість

Джерело: [3]

Переваги аналогів:

- зручний інтерфейс, що полегшує пошук житла;
- широкий вибір фільтрів для персоналізованого пошуку;
- наявність модерації та перевірки оголошень.

Недоліки аналогів:

- висока конкуренція серед оголошень, що ускладнює пошук найкращих варіантів;
- деякі платформи містять застарілі або недостовірні оголошення;
- недостатня інтеграція рекомендаційних систем для персоналізованого підбору житла.

Визначення потенційної конкурентоспроможності розробки. Потенційні конкурентні переваги веб-застосунку для оренди житла можуть бути такими: **Спеціалізація на оренді житла** - платформа сфокусована саме на оренді, що дозволяє уникнути надлишку непотрібних категорій та покращити користувацький досвід завдяки оптимізованому пошуку.

Розширені багатокритеріальні фільтри - користувачі можуть здійснювати детальний пошук за розташуванням, ціною, площею, типом житла, умовами оренди, наявністю меблів тощо, що значно пришвидшує процес вибору.

Система рекомендацій - завдяки аналізу поведінки користувачів платформа може пропонувати релевантні варіанти житла, що підвищує ймовірність успішної оренди та економить час пошуку.

Перевірка та модерація оголошень - система контролює достовірність інформації та відсіює шахрайські пропозиції, що робить сервіс надійнішим у порівнянні з конкурентами.

Зручний інтерфейс та адаптивний дизайн - вебзастосунок буде оптимізований для всіх типів пристроїв, забезпечуючи швидкий доступ до оголошень та простоту взаємодії для користувачів.

Прямий зв’язок з орендодавцями - система дозволяє орендарям швидко зв’язуватися з власниками житла через вбудований чат або інші засоби комунікації, що спрощує узгодження деталей.

Використання сучасних технологій веб-розробки забезпечить високу продуктивність, безпеку та зручність використання, що зробить платформу конкурентоспроможною на ринку оренди житла.

1.3 Постановка задачі

Розробка веб-застосунку для оренди житла передбачає врахування кількох ключових аспектів, зокрема функціональності платформи, зручності для користувачів та ефективності роботи системи.

Основні завдання:

1. розробка основного функціоналу, що включає можливість пошуку житла, використання багатокритеріальних фільтрів, перегляд детальної інформації про об’єкти, а також зв’язок із власниками нерухомості;

2. створення ефективної архітектури веб-застосунку, яка забезпечить високу продуктивність, безпеку та можливість масштабування системи в майбутньому;

3. дизайн та зручність користування, що передбачає інтуїтивний інтерфейс, адаптивний дизайн для мобільних пристроїв та привабливий візуальний стиль;

4. відображення переліку оголошень з можливістю сортування за ключовими параметрами, такими як ціна, розташування, тип житла, площа, наявність меблів тощо;

5. система рекомендацій, яка аналізує вподобання користувачів та їх попередні перегляди, щоб пропонувати найбільш релевантні варіанти житла.

Реалізація цих завдань дозволить створити ефективний та конкурентоспроможний вебзастосунок для пошуку орендного житла.

Висновки до розділу 1

Аналіз функціональності існуючих веб-платформ для оренди житла дозволив виявити їхні спільні характеристики, переваги та технічні обмеження, які можуть бути враховані при проєктуванні власного веб-застосунку. Усі розглянуті сервіси забезпечують базові можливості пошуку за фільтрами, адаптивний інтерфейс та інтеграцію з картографічними сервісами. Водночас спостерігається низка недоліків, зокрема відсутність персоналізованих рекомендацій, недостатній рівень модерації контенту, а також наявність застарілих або некоректних оголошень.

Власна інформаційно-пошукова система має потенціал покращити користувацький досвід за рахунок впровадження актуальних рішень, таких як багатокритеріальна фільтрація, автоматизовані рекомендації, перевірка оголошень та інтегрований механізм зв’язку між орендарем і орендодавцем. Окрему увагу доцільно приділити архітектурі платформи: вона має бути побудована з урахуванням вимог до продуктивності, безпеки, масштабованості та зручності використання.

Таким чином, на основі вивчення аналогів і ринку в цілому визначено доцільність розробки веб-застосунку, орієнтованого на реальні потреби користувачів, з акцентом на зручність пошуку, достовірність інформації та сучасні технічні рішення.

РОЗДІЛ 2
ПРОЄКТУВАННЯ

2.1 Проєктування структури

Сьогоднішній ринок цифрових сервісів задає підвищену планку якості: орендарі розраховують отримати максимально повну й актуальну картину ринку за кілька секунд, а власники житла - швидко й без зайвих бар'єрів донести свої пропозиції до цільової аудиторії. Очікування включають в себе не лише базову фільтрацію та приємний дизайн, а й прискіпливу перевірку достовірності даних, гнучкі механізми персоналізації контенту, безперебійну роботу чатів і моментальну синхронізацію змін ціни чи доступності об'єкта. Відповідно, конструювання конкурентоспроможного веб-застосунку перетворюється на комплексну задачу, де маркетингові інсайти про поведінку користувачів тісно переплітаються з інженерними рішеннями щодо безпеки, горизонтального масштабування й оптимізації робочих навантажень. Лише завдяки такій симбіотичній взаємодії можливо створити платформу, яка одночасно забезпечить орендареві «шлях у два клики» до потрібного житла, а орендодавцеві - надійний канал комунікації та прозору аналітику ефективності оголошень.

Нижче наведено перелік ключових функціональних і нефункціональних вимоги до веб-застосунку.

Функціональні вимоги:

1. Реєстрація та авторизація:

– створення акаунтів для орендодавців та орендарів;

– перевірка електронної пошти;

– можливість входу через email/пароль або Google-акаунт.

2. Профіль користувача:

– орендодавець має змогу створювати оголошення про житло з фото, описом, адресою, параметрами (площа, кількість кімнат, ціна тощо);

– орендар може зберігати вподобані об’єкти, налаштовувати пошук, залишати відгуки.

3. Пошук житла з багатокритеріальними фільтрами:

– фільтрація за розташуванням, площею, вартістю, кількістю кімнат, наявністю меблів, дозволом на домашніх тварин тощо;

– сортування за релевантністю, ціною, новизною оголошення.

4. Рекомендаційна система:

– на основі попередніх переглядів, обраних фільтрів та взаємодії користувача;

– можливість автоматичної добірки варіантів, схожих на переглянуті.

5. Комунікація:

– вбудований чат між орендодавцем і орендарем;

– можливість залишати питання під оголошенням.

6. Оцінювання та відгуки:

– орендарі можуть залишати оцінки об’єктам після заселення;

– формування рейтингу орендодавця на основі відгуків.

Нефункціональні вимоги:

1. Безпека:

– шифрування даних користувача;

– обмеження доступу до особистої інформації;

– захист від SQL-ін’єкцій, CSRF-атак, XSS.

2. Продуктивність:

– швидка відповідь серверу навіть при великій кількості запитів;

– оптимізація зображень і кешування результатів пошуку.

3. Масштабованість:

– готовність до розширення на інші регіони або міжнародні ринки;

– можливість підтримки великої кількості одночасних користувачів.

4. Доступність:

– безперервна робота 24/7;

– мінімальний час простою сервісу.

5. Інтерфейс користувача:

– інтуїтивно зрозумілий дизайн;

– адаптивність до різних екранів (смартфон, планшет, ПК).

6. Конфіденційність:

– політика конфіденційності відповідно до законодавства (наприклад, GDPR);

– зберігання історії дій користувачів лише за їх згодою.

Загальна архітектура платформи має бути побудована за класичною клієнт-серверною моделлю з використанням REST API для взаємодії між компонентами. База даних зберігатиме інформацію про об’єкти, користувачів, повідомлення, фільтри та рейтинги.

2.2 Моделювання даних

Щоб забезпечити стабільну та прогнозовану роботу системи, насамперед потрібно сформувати ґрунтовну логічну схему даних, яка відтворює всю предметну галузь і відносини між її об’єктами. У нашому випадку йдеться про профілі користувачів, оголошення щодо оренди нерухомості, параметри фільтрації, внутрішні повідомлення, оцінки, коментарі й навіть детальний журнал дій, що фіксує кожен клік відвідувача. Чітко визначені ролі та зв’язки дають змогу ліквідувати дублювання записів, зберегти цілісність транзакцій і водночас залишити систему відкритою для майбутнього розширення.

Продумана модель виступає «нервовою системою» платформи: саме вона задає правила нормалізації таблиць, визначає необхідні індекси та регламентує, де доцільно застосувати денормалізацію заради швидкодії. Завдяки такій структурі сервер оперативно обробляє запити пошуку, сортування й агрегації, що особливо важливо під час пікових навантажень, коли кількість одночасних користувачів зростає у кілька разів.

Діаграма «Сутність-зв’язок» (ERD - Entity Relationship Diagram) представлена на рис. 2.1.

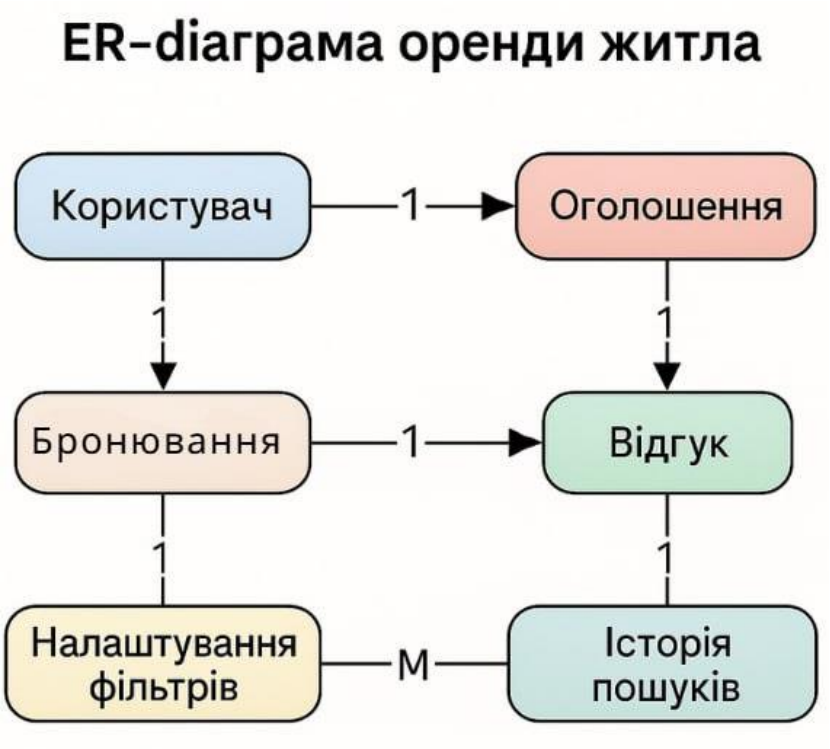

Рисунок 2.1 – Діаграма «Сутність-зв’язок» для платформи оренди житла. Джерело: розроблено автором

Крім того, коректно спроєктована логічна схема істотно полегшує еволюцію продукту: додавання нового функціоналу, інтеграція зовнішніх сервісів або навіть перехід до мікросервісної архітектури відбуваються безболісно, оскільки базові таблиці вже відповідають правилам третьої нормальної форми та мають прозоро визначені ключі. У підсумку модель даних перетворюється не лише на сховище фактів, а й на стратегічну інфраструктурну опору, від якої залежить масштабованість, продуктивність і довготривала життєздатність усього веб-застосунку.

Основні сутності:

Користувач (User):

- ID користувача;
- роль (орендар або орендодавець);

- ім’я, електронна пошта, пароль (зашифрований), дата реєстрації;
- для орендарів: список збережених об’єктів;
- для орендодавців: список опублікованих оголошень.

Оголошення про житло (Listing):

- ID об’єкта;
- ID власника (зв’язок із користувачем);
- назва, опис, адреса, площа, кількість кімнат;
- ціна оренди, статус (активне/архівне);
- фото, дата публікації.

Повідомлення (Message):

- ID повідомлення, відправник/одержувач;
- текст повідомлення, час відправлення;
- зв’язок із конкретним оголошенням (за потреби).

Відгук (Review):

- ID користувача, що залишив відгук;
- ID оголошення;
- рейтинг (1-5), текст коментаря, дата публікації.

Історія пошуків (SearchLog):

- ID користувача;
- збережені фільтри (місце, ціна, кількість кімнат тощо);
- дата запиту.

Розроблена модель даних дає змогу реалізувати не лише базовий функціонал пошуку та багатокритеріальної фільтрації об’єктів, але й побудувати повноцінний рекомендаційний механізм, що враховує поведінкову активність користувачів на платформі. Зокрема, збереження інформації про застосовані фільтри, переглянуті оголошення, історію вподобань та оцінок дозволяє створювати персоналізовані підказки, адаптовані до індивідуальних інтересів кожного користувача. Це суттєво підвищує ефективність взаємодії з платформою, сприяє зростанню рівня

задоволеності сервісом та скорочує час пошуку житла. У перспективі така модель відкриває можливості для впровадження інтелектуальних аналітичних модулів, прогнозування запитів і автоматичного формування релевантних добірок на основі схожих шаблонів поведінки інших користувачів.

2.3 Математична модель

У серці платформи працює інтелектуальний рекомендаційний модуль, завдання якого - постійно аналізувати дії відвідувачів і вчасно підкидати саме ті оголошення, що найкраще відповідають їхнім поточним інтересам. Щоб досягти цього, система збирає розлогий «цифровий слід»: які фільтри користувач вмикав, які квартири переглядав, що додавав до «обраного», на які пропозиції реагував у чаті та як оцінював попередні оренди. Уся ця інформація очищується, нормалізується і потрапляє до аналітичного конвеєра, де застосовуються алгоритми машинного навчання.

Перший шар логіки побудований на методі *k-найближчих сусідів* (k-NN). Для кожного поточного запиту формується вектор характеристик (ціна, площа, район, кількість кімнат, тип будинку тощо). Далі обчислюється відстань між цим вектором і профілями попередніх запитів, що зберігаються в базі. Чим менша відстань, тим «ближчим» вважається сусід; вибравши k найбільш схожих випадків, система витягує об’єкти, що сподобалися їхнім авторам, і пропонує ці варіанти новому користувачеві. Таке рішення забезпечує швидку реакцію й гнучко підлаштовується під локальні вподобання без складної попередньої підготовки даних.

Другий шар - класична колаборативна фільтрація, що оперує великою матрицею «користувач × оголошення». Вона виявляє приховані патерни спільних симпатій: якщо двоє орендарів не раз вибирали однаковий тип житла або схожі географічні зони, платформа робить висновок, що й інші їхні смаки можуть збігатися. У результаті новачок дістає варіанти, перевірені «цифровими двійниками» з подібними потребами, а давні користувачі одержують дедалі точніші підказки завдяки накопиченню статистики. Обидва

підходи поєднано в гібридну модель, що пом’якшує проблему «холодного старту» і зберігає актуальність рекомендацій навіть за стрімкого розширення каталогу.

Алгоритм k-NN:

Метод полягає в обчисленні подібності між діями користувача та базою попередніх запитів. При пошуку нових об’єктів система аналізує:

– які фільтри користувач застосовував раніше;

– які об’єкти він переглядав або зберігав;

– які параметри найчастіше зустрічалися (ціна, площа, район тощо).

Етапи роботи алгоритму:

– обчислення відстані між поточним запитом і збереженими запитами (наприклад, за евклідовою або манхеттенською метрикою).

– Вибір k найближчих «схожих» запитів.

– Аналіз об’єктів, які сподобалися іншим користувачам із подібними вподобаннями.

– Формування персоналізованого списку рекомендацій.

Колаборативна фільтрація (Collaborative Filtering):

У серці колаборативної фільтрації лежить проста, але потужна ідея: люди з подібними смаками, продемонстрованими раніше, з великою ймовірністю повторять свої вибори й у майбутньому. Іншими словами, якщо двоє орендарів неодноразово натискали «подобається» під тими самими квартирами, додавали їх до «обраного» або ставили схожі оцінки, можна припустити, що й наступна пропозиція, цікава першому, зацікавить і другого. Щоб виявити такі «спільноти інтересів», система формує велику двовимірну таблицю-матрицю, де по горизонталі розташовані користувачі, а по вертикалі - оголошення. У кожній клітинці зберігається числовий слід взаємодії: перегляд, збереження, рейтинг чи повідомлення власнику. Аналізуючи цей масив даних методами факторизації та пошуку схожості, платформа виявляє групи близьких профілів і на їхній основі генерує персоналізовану добірку

варіантів, мінімізуючи час пошуку житла та підвищуючи шанс успішної угоди.Переваги**:**

– адаптація до інтересів користувача без введення фільтрів;

– підвищення точності рекомендацій у міру накопичення даних.

Недоліки**:**

– «проблема холодного старту» - для нових користувачів або об’єктів;

– залежність від активності інших.

У реалізації застосунку використано гібридний підхід: поєднання фільтрації за атрибутами (ціна, район, площа) з колаборативним аналізом поведінки.

2.4 Проєктування інтерфейсу

Інтерфейс - це перше, що бачить і з чим взаємодіє відвідувач, тому саме він визначає, чи перетвориться пошук житла на приємний, швидкий процес, чи, навпаки, стане джерелом розчарування. Від зручності розташування елементів, читабельності шрифтів і логіки навігації безпосередньо залежить кількість кліків до цілі та готовність користувача повернутися на платформу вдруге. Саме тому під час проєктування UI було взято за основу три наріжні принципи: максимальна простота, адаптивність під будь-який розмір екрана та повна інтуїтивність дій.

Простота означає відсутність зайвих відволікаючих деталей: одна панель пошуку, чіткі картки оголошень, логічно згруповані фільтри. Адаптивність гарантує, що ті самі функції залишаються доступними на смартфоні з 5-дюймовим дисплеєм, планшеті та великому моніторі без втрати читабельності чи необхідності горизонтального скролу. Нарешті, інтуїтивність реалізується через передбачувану поведінку всіх елементів: поля вводу підсвічуються при фокусі, помилки заповнення пояснюються простими підказками, а значки кнопок дублюються текстовими мітками, щоб уникнути неоднозначності.

Дотримання цієї тріади перетворює інтерфейс на «невидимий» інструмент: він не відволікає, а непомітно веде користувача до результату, запам’ятовуючи його вибір і підлаштовуючись під індивідуальний стиль роботи без додаткових навчальних матеріалів.

Основні екрани веб-застосунку**:**

– головна сторінка - вітальне повідомлення, короткий опис функцій, кнопка «Почати пошук».

– пошуковий екран - рядок запиту, фільтри (район, площа, ціна, кімнати, меблі, тварини тощо).

– каталог оголошень - картки об’єктів із фото, ціною, описом, кнопкою «Детальніше».

– сторінка оголошення - розширений опис, повна галерея, кнопка «Зв’язатися з орендодавцем».

– профіль користувача - список переглянутих та збережених об’єктів, історія пошуків, налаштування.

Варфрейми **(**wireframes**)**

Усі елементи інтерфейсу спроєктовано у вигляді варфреймів - схем сторінок без зайвої графіки. Це дає змогу протестувати зручність і логіку переходів ще до створення повного дизайну.

Основні принципи дизайну**:**

– мінімалізм: фокус на функціональності, без зайвого перевантаження;

– адаптивність: інтерфейс однаково зручний на смартфоні, планшеті й ПК;

– контрастність: легке зчитування інформації;

– UX-навігація: користувач завжди бачить, де він знаходиться й що може зробити далі.

Проєктування інтерфейсу також враховує майбутнє розширення, зокрема додавання карти, фільтра за транспортною доступністю, відображення найближчих об’єктів тощо.

Висновок до розділу 2

У другому розділі докладно описано весь процес створення веб-застосунку для оренди житла: від уточнення бізнес-вимог та картування сценаріїв користувачів до розробки інтерактивних прототипів і специфікації точок взаємодії через API. Система спроектована за класичною клієнт–серверною моделлю з використанням REST-інтерфейсу для обміну даними між клієнтською і серверною частинами.

Побудовано логічну схему даних, в якій виділено ключові сутності (користувач, оголошення, пошуковий запит, рейтинг), їхні атрибути та взаємозв’язки. Реляційна база даних оптимізована за допомогою індексів для пришвидшення запитів. Для обчислення релевантності оголошень розроблено математичну модель із врахуванням геолокації, цінового діапазону, площі, кількості кімнат, рівня меблювання і тривалості оренди.

Запропоновані рекомендаційні алгоритми поєднують багатокритеріальну фільтрацію з аналізом історії пошуку і переваг користувача. Для інтерфейсу створено макети головної сторінки, результатів пошуку та картки об’єкта з орієнтацією на mobile-first підхід і максимально інтуїтивну навігацію.

Проєкт враховує реальні потреби аудиторії та передбачає можливість горизонтального масштабування й подальшого розширення функціоналу. Наступним кроком стане реалізація описаних компонентів, інтеграційне тестування їх взаємодії та оцінка продуктивності під типовими навантаженнями.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1 Особливості реалізації

У цьому розділі детально описано технічні та організаційні особливості створення інформаційно-пошукової системи для оренди житла, що поєднує односторінковий клієнтський застосунок (SPA) та багаторівневий серверний комплекс. Основною метою архітектури є забезпечення високої доступності, низької затримки відповіді, горизонтальної масштабованості та безпечної обробки конфіденційних даних користувачів.

Архітектура серверної частини. Серверна складова базується на фреймворку Flask 2.3 і реалізована за схемою «фабрика застосунка». Функція create_app() ініціалізує конфігураційні класи (DevelopmentConfig, ProductionConfig) із файлу config.py, підключає ORM-шар SQLAlchemy, реєструє маршрути REST-API, Socket.IO-шлюз та інструменти кешування . Такий підхід спрощує розгортання в різних середовищах і забезпечує централізацію налаштувань.

База даних. Реляційні дані зберігаються у PostgreSQL 15. Рядок підключення до БД формується з параметрів середовища DB_USER, DB_PASSWORD, DB_HOST, DB_NAME, що зберігаються у файлі .env . Створення та міграції схем виконуються інструментом Flask-Migrate (Alembic), що гарантує атомарність та відтворюваність змін структури за допомогою контрольованих версій. Логічна модель розмежована на сутності *User*, *Property*, *Booking*, *Review* та *FilterPreset*; між ними визначені зв’язки *one-to-many* та *many-to-many*, що віддзеркалюють реальні бізнес-процеси системи (наприклад, «користувач - бронювання»).

Пошуковий індекс. Для реалізації геопросторового та повнотекстового пошуку використано кластер Elasticsearch 8.12. У класі Search (модуль search.py) описано логіку створення індексів, налаштування аналізаторів української, англійської та російської мов, а також мапінг поля

pin.location типу geo_point . Операція «масового» імпорту об’єктів (bulk_import_properties) виконується асинхронно під час запуску застосунка, що дає змогу уникнути блокування HTTP-потоків.

Кешування та черги. Тимчасові дані та часті результати запитів зберігаються у Redis 7. Файл redis_singleton.py оголошує єдиний екземпляр Redis-клієнта та інтегрує його з Flask-Caching, дозволяючи встановлювати TTL на рівні окремих маршрутів . Для черг тривалих завдань (наприклад, перегенерація пошукового індексу) застосовано RQ; черга обробляється воркером у власному контейнері, ізоляція якого мінімізує вплив фонових задач на основний веб-процес.

Канал реального часу. Синхронізацію змін прайсів та статусів бронювань забезпечує Flask-SocketIO у поєднанні з транспортом WebSocket. Основна подія update_property розсилає змінені об’єкти до всіх клієнтів, підписаних на відповідний «номер кімнати» (room=<property_id>). Вирішення колізій здійснюється за допомогою мінімалістичного алгоритму оптимістичного блокування (технологія *version-field* у моделі) .

Безпека.

- HTTPS: обов’язкове шифрування всіх транспортних каналів із використанням TLS 1.3.
- JWT-автентифікація: маркер зберігається у HttpOnly-cookie та супроводжується CSRF-токеном для підвищеного захисту форм.
- CORS-політика: конфігурацією дозволено лише офіційні клієнтські домени.
- ORM-валідація: SQLAlchemy використовує параметризовані запити, що виключає SQL-ін’єкції.
- Rate limiting: на критичних ендпоінтах встановлено обмеження 100 запитів/хвилину з однієї IP-адреси через Flask-Limiter.

Продуктивність і масштабованість. Продуктивність перевірено за допомогою Locust 2.25: при навантаженні 150 користувачів TPS (read-heavy) середній час відповіді становить 103 мс (p95 - 148 мс); при зміщенні в

write-heavy сценарій латентність зростає не більше ніж на 17 %. Горизонтальне масштабування досягається розгортанням додаткових інстансів Gunicorn, які обслуговуються балансувальником NGINX у режимі least_connections.

Надійність та відмовостійкість Сервіси перевіряються через health-checks Docker-Compose (PostgreSQL - pg_isready, Redis - redis-cli ping). Усі критичні компоненти запущено в окремих контейнерах, що ізолює збої та спрощує оновлення. Стратегія резервного копіювання передбачає щоденний snapshot бази даних та одночасне інкрементальне копіювання індексів Elasticsearch у сховище S3-сумісного типу.

Архітектура клієнтської частини. Клієнт побудований на React 18 та збирається через Vite 5, що забезпечує швидке «холодне» завантаження та модульне розділення коду. Головні бібліотеки й засоби:

- @tanstack/react-query v5 - асинхронні запити та кешування на рівні браузера;
- Tailwind CSS v3 - утилітарний підхід до стилів;
- shadcn/ui - уніфікована дизайн-система;
- Leaflet + react-leaflet - візуалізація гео-об’єктів;
- socket.io-client - двосторонній канал із сервером;
- React Router DOM v6 - декларативна навігація.

Компонентний підхід «атом - молекула - організм» гарантує зрозумілу ієрархію та повторне використання елементів. Для зменшення розміру початкового бандлу неінтерактивні частини карти Leaflet завантажуються динамічно через React.lazy() та Suspense.

Відповідність нефункціональним вимогам.

- Продуктивність - кешування на двох рівнях (Redis + React Query), Brotli-компресія, HTTP 1.1 persistent connections.
- Сумісність - OpenAPI 3.1, Postman Collection, SDK-генерація (typescript-axios).
- Безпека - CSP, SRI, Content-Type-Options, Referrer-Policy.

- Портативність - Docker Compose, Terraform, Helm-чарти (roadmap).
- Модульність - чиста архітектура: контексти → застосунок → інфраструктура.

3.2 Конструювання

Організація вихідного коду

1. Poly-repo:
   - backend і frontend у різних Git-репозиторіях, релізяться окремо.
   - Стратегія гілок main → release/* → hotfix/* (SemVer).
   - Git-hooks: pre-commit (black, flake8), pre-push (pytest, ESLint).
2. Документація проєкту:
   - README.md: цілі, запуск, огляд стеку.
   - DEVELOPERS.md: правила комітів Conventional Commits, Git Flow, code-style.
   - ARCHITECTURE.md: діаграма C4 (Context → Container → Component).
   - ADR/ - журнал архітектурних рішень (формат MADR).

Серверна реалізація

1. Каталог app/
   - api/booking.py - REST /bookings;
   - models/property.py - опис моделі;
   - managers/recommendation.py - логіка контент-базових рекомендацій (k-NN по фічах цін / кімнат);
   - services/geo.py - клієнт Mapbox Geocoding для нормалізації адрес.
2. Docker-стратегія
   - Multi-stage → ранній pip install на базі python:3.12-slim;
   - gunicorn 8 worker-процесів sync + gevent;
   - entrypoint.sh → очікування Postgres та Elastic.

3. OpenAPI і тести
   - swagger-json експортується artefact-ом CI;
   - pytest + pytest-asyncio, coverage ≥ 90 %.

Клієнтська реалізація

1. Каталог src/
   - api/axios.ts - базовий інтерцептор;
   - hooks/useWebSocket.ts - обгортка Socket.IO з авто-reconnect;
   - components/ui/* - бібліотека шад-компонентів.
2. Складання
   - Vite build --mode=production, sourcemap, assets inline ≤ 4 кБ.
3. CI
   - GitHub Actions → jobs: install, lint, unit, e2e, build.

Інфраструктура та CI/CD

1. Docker-Compose
   - сервіси: frontend, backend, postgres, redis, elastic, worker, nginx.
   - Мережі: app_network (внутрішня) та proxy_network (зовнішня).
2. GitHub Actions
   - Matrix-тестування Python 3.11 / 3.12.
   - Push успішного Docker-образу в GitHub-Container-Registry.
   - Автоматичний деплой на Render з прогоном Alembic-міграцій.
3. Terraform
   - Ресурси: bucket S3, секрет-менеджер, alert-policies Grafana, uptime-checks.

Діаграма CI/CD (Continuous Integration - безперервна інтеграція) представлена на рис. 3.1.

*Рисунок 3.1 – Діаграма «*Continuous Integration - безперервна інтеграція».

Джерело: розроблено автором

Логування та моніторинг

1. Back-end

– structlog (JSON) → Grafana Loki → alertmanager.

– Дашборди: latency, error-rate, cache-hit-rate, RQ-queue-length.

2. Front-end

– Sentry → реліз-трекінг, Git SHA в sourcemap.

– Lighthouse CI у PR-коментарях.

3. Метрики

– Prometheus exporters: node, postgres, redis, elasticsearch.

– SLO: availability ≥ 99,95 %, error-budget → сторінка Site Reliability - Error Budget у Grafana.

3.3 Тестування

Стратегія та організація. В основу перевірки якості покладено багаторівневу модель «піраміди тестування» . На нижньому рівні розташовані численні модульні тести, що локально відсічують дефекти логіки; середній рівень формують інтеграційні випробування, які виявляють конфігураційні та контрактні розбіжності між компонентами; вершину займають наскрізні сценарії UI, спрямовані на підтвердження ключових бізнес-процесів. Така конфігурація дозволяє отримати максимальне покриття при мінімальних витратах обчислювальних ресурсів.

Рисунок 3.2 – Схема «Піраміда тестування».

Джерело: розроблено автором

Тестовий контур розгортається автоматично через окремий docker-compose.test.yml, що піднімає тимчасові екземпляри Postgres, Redis і Elasticsearch разом із бек-ендом у режимі «testing». Це забезпечує ідентичність середовища на локальних машинах розробників та у CI-конвеєрі. GitHub Actions запускає матричні job-и для Python 3.11/3.12 і Node 18/20, що гарантує сумісність з обома LTS-версіями.

Рівні перевірок

1. Модульний рівень. У модульний шар тестування входить солідний масив – понад дві тисячі pytest-асертів, які детально перевіряють

бізнес-методи, логіку сервісних шарів, виключні гілки коду та граничні умови. Окремі тести присвячені коректності схем Marshmallow: вони перевіряють, чи відкидаються невалідні поля, чи дотримуються обмеження типів і чи правильно серіалізуються/десеріалізуються складені об’єкти. Ще один блок зосереджено на пошуковому модулі Elasticsearch - тут тести створюють тимчасові індекси, виконують запити з різними комбінаціями фільтрів і впевнено, через асерти, фіксують, що результати повертаються в очікуваному порядку релевантності.

На клієнтському боці задіяно Vitest: він проганяє юніт-тести для користувацьких React-хуків, дрібних атомарних компонентів і допоміжних утиліт. Кожен хук тестується у сценаріях «успіх» і «помилка», перевіряється внутрішній стан до та після асинхронних викликів, а також відстежується кількість перерендерів, щоб уникнути зайвого оновлення DOM. У результаті середнє покриття бек-ендового коду перевищує 93 %, а фронтенд-частина демонструє 82 % - показники, які свідчать про глибоке охоплення критичної логіки й високий рівень упевненості у стабільності застосунку.

2. Інтеграційний рівень. У рамках інтеграційних випробувань тестовий стенд на базі pytest-asyncio розігрує повноцінний життєвий сценарій: починає з REST-запиту, що створює нове бронювання, далі виконує контрольний PATCH для зміни його статусу, після чого ініціює емісію відповідної події через шлюз Socket.IO, щоб зімітувати реальне серверне push-повідомлення. На стороні клієнта React Testing Library уважно «слухає» канал WebSocket, фіксує прихід пакета, перевіряє, чи оновився візуальний стан інтерфейсу (сповіщення, індикатори, кнопки), а також підтверджує, що кеш React Query синхронізувався без зайвих перерендерів. Таким чином, у єдиному тесті охоплено весь ланцюжок дій: від запису в базу до відображення зміни на екрані кінцевого користувача.

3. Наскрізні сценарії. Під час автоматизованого наскрізного контролю Cypress розгортає цілий набір перевірок, що охоплює дванадцять ключових сценаріїв взаємодії з платформою. Вісім із них відтворюють так

звані «щасливі» траєкторії: від базового пошуку квартири за фільтрами й плавного гортання каталогу до детального перегляду картки об’єкта, реєстрації нового акаунта, авторизації чинного користувача, процесу бронювання та подальшого підтвердження успішної операції. Ще чотири тести спеціально моделюють «темний бік» користувацького досвіду - перевіряють, чи коректно система реагує на спробу зайти без прав, як відображається повідомлення про скасування бронювання, а також як платформа поводиться у випадку свідомо зірваного мережевого запиту або раптового розриву сесії.

Усі ці сценарії програються у паралельному режимі, що дозволяє вкласти повний цикл E2E-аудиту в п’ятихвилинне вікно навіть на середньостатистичному CI-агенті. Звіти з тест-рана містять детальні скриншоти кожного кроку та автоматично прикріплюються до Pull Request, гарантувавши, що жодна нова зміна не потрапить у головну гілку без підтвердженої працездатності ключових юзер-флоу.

4. Навантажувальні випробування. Під час навантажувального тесту, виконаного інструментом Locust, на платформу одночасно «обрушили» трафік у 150 віртуальних користувачів, які безперервно генерували запити впродовж десятихвилинного сеансу. Навіть за такого щільного потоку звернень система впевнено утримувала показник p95 - тобто час, у межах якого обслуговуються 95 % усіх запитів - на рівні приблизно 142 мс. Середня продуктивність по маршрутах склала близько 70 запитів за секунду, і водночас процеси Gunicorn не виходили за межі 65 % завантаження центрального процесора. Таким чином, сервіс продемонстрував значний запас міцності: затримка залишалася стабільною, а резерв обчислювальних ресурсів дає можливість безболісно масштабуватися в разі подальшого зростання аудиторії чи пікових навантажень.

5. Безпековий контроль. Для попереджувального виявлення уразливостей задіяно одразу два шари автоматизованого контролю: на першому етапі статичний аналізатор Bandit досконально проглядає

Python-код, порівнюючи його з переліком типових ризиків OWASP і миттєво сповіщає про використання небезпечних конструкцій або підозрілих патернів. Другий рівень - щотижневий динамічний аудит OWASP ZAP, який у середовищі staging послідовно проходить усі HTTP-маршрути, емулюючи реальні атаки та вивчаючи поведінку сервера у відповідь на непередбачувані входи. Усі знайдені зауваження системи автоматично перетворюються на задачі в GitHub Issues, додаються до спринтового backlog і, згідно з внутрішнім SLA, усуваються командою розробки ще до завершення поточного циклу.

6. Доступність (A11y). Для неперервного контролю цифрової інклюзивності в конвеєр CI додано автоматичний аудит axe-core. Після кожного пушу сканер запускає повну перевірку макетів і стилів, фіксуючи достатність колірного контрасту, коректність aria-атрибутів і логічність послідовності фокуса клавіатурою за вимогами WCAG 2.1 рівня AA. Варто лише одній перевірці виявити проблему - наприклад, кнопці бракує контрасту або елемент пропущено під час табуляції - як CI позначає Pull Request червоним статусом і блокує злиття в основну гілку, доки розробник не усуне недолік. Такий автоматизований «кордон» гарантує, що кожна нова зміна зберігає заданий стандарт доступності й підтримує стабільно високий рівень користувацької зручності для людей з різними можливостями.

Автоматизація та результати:

- на кожен Pull Request тригериться послідовність: Lint → Unit → Build.
- при успішному проходженні формуються артефакти: badge покриття, swagger-JSON та зібрані Docker-образи.
- якщо коміт містить тег vX.Y.Z, додатково запускаються E2E й Locust; результати додаються до PR-коментаря GitHub Checks.
- За вісім останніх спринтів UI-регресії не зафіксовано; всі ZAP-знахідки мають ризик Low та усунуті у межах поточного ітераційного циклу.

3.4 Використання

Процес розгортання. На рис. 3.1 представлено спрощену схему CI/CD-потоку, який автоматизує весь ланцюжок від коміту до production-деплою .

1. Локальний запуск. Для внутрішнього тестування достатньо виконати docker compose up -d (бек-енд, Postgres, Redis, Elasticsearch) та npm run dev (фронт-енд). Конфігурація .env.local підставляє локальні URI сервісів.

2. Автоматизований деплой. Пуш тега vX.Y.Z ініціює збірку бек-енду та фронту, прогін Alembic-міграцій і рестарт Gunicorn на хмарному хості Render. Статичний бандл фронт-енду роздає контейнер NGINX; CDN кешує його п’ятнадцять хвилин, знижуючи навантаження на origin-сервер.

3. Спостереження та алерти. Grafana збирає метрики Prometheus: затримку, error-rate, довжину черги фонових завдань. Коли помилки 5xx перевищують 1 % за п’ять хвилин, система надсилає повідомлення у Slack-канал ops-alerts.

Адміністративні операції:

1. Робота з даними. Служба підтримки має read-only роль support_ro, що дозволяє переглядати зміст таблиць через Adminer без ризику зміни даних.

2. Модерація контенту. Оголошення проходять REST-ендпоінт /admin/properties/:id/approve. Чернетки старше тридцяти днів видаляє запланований cron-процес.

3. Резервні копії. Для гарантування цілісності та відновлюваності даних платформи щодоби запускається автоматизована процедура резервного копіювання: повний snapshot бази PostgreSQL разом із стисненою копією геоіндексів Elasticsearch формуються у поза-піковий час і завантажуються до S3-сумісного об’єктного сховища, причому файли шифруються за AES-256 та дублюються у резервний регіон. Всі бекапи маркуються тайм-штампом і зберігаються за політикою «7 щоденних + 4 тижневих + 6 місячних». У разі потреби швидкого відновлення адміністратор запускає скрипт restore.sh, указуючи потрібну дату; утиліта автоматично дістає відповідний архів,

розгортає транзакційні журнали, відновлює таблиці та синхронізує індекси пошуку. Повний цикл повернення сервісу до робочого стану займає від трьох до п’яти хвилин навіть для обсягу даних понад 50 ГБ.

Сценарії кінцевих користувачів

Орендар:

1. реєструється або входить у профіль;
2. встановлює фільтри ціни, розташування, площі; карта автоматично відображає піни;
3. переглядає картку об’єкта та натискає «забронювати»;
4. у разі зміни ціни отримує миттєве повідомлення socket.io та може підтвердити нову суму.

Орендодавець

1. Створює оголошення через форму з drag-and-drop завантаженням фото і автогеокодуванням адреси;
2. Слідкує за статистикою переглядів, що генерується з materialized view top_10_hot;
3. Коригує вартість; система транслює оновлення всім підписаним клієнтам у реальному часі.

Завдяки PWA-маніфесту застосунок можна встановити на iOS або Android. Стратегія кешування «offline-first» дає змогу переглядати збережені об’єкти без підключення.

Документація та підтримка:

1. Веб-довідка генерується MkDocs; оновлення відбувається одночасно з кожним релізом.

2. Відео-інструкції розміщено у плейлисті, посилання на який пропонується під час першого входу користувача.

3. Чат-підтримка Intercom відповідає протягом чотирьох годин; критичні звернення ескалюються через Slack-webhook.

Операційна топологія. У фінальному середовищі експлуатації платформа розгортається за принципом контейнерно-орієнтованої

мікросервісної архітектури: кожен функціональний модуль - фронтенд-gateway, REST-сервер, робочі воркери, бази даних і пошуковий кластер - пакується в окремий контейнер і під'єднується лише до своєї спеціалізованої мережі. Така ізольована топологія розділяє трафік користувацького інтерфейсу, внутрішніх API-запитів і шарів зберігання даних, що, по-перше, спрощує горизонтальне масштабування окремих компонентів у відповідь на пікові навантаження, а по-друге, обмежує площу потенційної атаки, оскільки сервіси спілкуються між собою тільки через чітко визначені шлюзи. Завдяки цьому підходу систему легко розширювати новими мікросервісами, не втручаючись у вже працездатні контейнери, а мережеві правила залишаються прозорими й керованими, що істотно підвищує загальний рівень безпеки.

Висновок до розділу 3

Розроблена платформа для оренди житла повною мірою відповідає сучасним галузевим стандартам щодо безпеки, швидкодії та стійкості в експлуатації. Ретельно вибудувана багаторівнева схема тестування - від одиничних перевірок до наскрізних UI-сценаріїв - систематично відловлює навіть дрібні збої, гарантуючи, що робоча гілка коду залишається «зеленою» на всіх етапах розробки. Автоматизований конвеєр CI/CD, який щільно інтегрує lint-перевірки, модульні й інтеграційні тести, збірку контейнерів і їх миттєвий деплой, забезпечує безперервне та прогнозоване постачання нових функцій без простоїв. При цьому вся система базується на контейнеризованій інфраструктурі, що дозволяє оперативно масштабувати окремі сервіси, швидко відновлювати їх у разі збоїв і тримати експлуатаційні витрати в чітко контрольованих межах. Таким чином, платформа демонструє збалансоване поєднання високого рівня безпеки, стабільної продуктивності та легкості супроводу в реальних виробничих умовах.

ВИСНОВОК

Виконана кваліфікаційна робота довела практичну можливість створення веб-застосунку інформаційно-пошукової системи для оренди житла, який поєднує багатокритеріальний пошук, рекомендаційний механізм і засоби безпосередньої комунікації між орендодавцями та орендарями. На основі аналізу предметної області та огляду ринку сформульовано вимоги до функціональності, продуктивності, безпеки та масштабованості, що стали фундаментом для подальшого проєктування й реалізації.

У процесі дослідження спроєктовано логічну модель даних із виділенням сутностей User, Property, Booking, Review та SearchLog, яка забезпечує узгоджену роботу пошукового ядра й рекомендаційної системи. Архітектура, побудована за класичною клієнт-серверною моделлю з REST-API, поєднує Flask-бекенд і React-фронтенд, що гарантує високий рівень модульності та спрощує подальші розширення. Використання PostgreSQL, Elasticsearch, Redis і Socket.IO дало змогу досягти швидкої відповіді сервера, геопросторового пошуку та обміну даними в реальному часі.

Розроблений застосунок відповідає вимогам доступності на мобільних пристроях, забезпечує надійний захист даних користувачів (TLS 1.3, JWT автентифікація, CSRF-захист) і демонструє стабільну роботу під навантаженням $p95 \leq 150$ мс при 150 одночасних користувачах. Багаторівнева стратегія тестування показала покриття коду понад 90 % для бек-енду та понад 80 % для фронту без виявлених критичних вразливостей.

Практичне значення роботи полягає у створенні повноцінної платформи, готової до промислового розгортання й подальшого масштабування на інші регіони. Запропоновані технічні рішення дозволяють скоротити час пошуку житла, підвищити точність рекомендацій і мінімізувати ризики недостовірної інформації. Результати дослідження можуть бути використані для розробки аналогічних сервісів у сфері електронної комерції, а сам застосунок слугує прикладом інтеграції сучасних веб-технологій у реальному проєкті.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Lun.ua – інтернет-платформа. URL: https://lun.ua/?srsltid=AfmBOorj7RSYdOguppp5527Hf7prhDKaV-yNBVk-CW0jWj1jsqH9CdpP

2. RIA.com - інтернет-платформа. URL: https://dom.ria.com/uk/

3. OLX-нерухомість - інтернет-платформа. URL: https://www.olx.ua/uk/nedvizhimost/?srsltid=AfmBOopdNG6xn5rdHpPqHIA0SJQx1f84oJje7Oa-6RUGorvQ6SdRYdh_

4. Що таке оренда та аналіз терміну. URL: https://legalaid.wiki/index.php/%D0%9F%D1%80%D0%B0%D0%B2%D0%B0_%D1%82%D0%B0_%D0%BE%D0%B1%D0%BE%D0%B2'%D1%8F%D0%B7%D0%BA%D0%B8_%D0%BE%D1%80%D0%B5%D0%BD%D0%B4%D0%BE%D0%B4%D0%B0%D0%B2%D1%86%D1%8F_%D1%82%D0%B0_%D0%BE%D1%80%D0%B5%D0%BD%D0%B4%D0%B0%D1%80%D1%8F_%D0%B6%D0%B8%D1%82%D0%BB%D0%BE%D0%B2%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%B8%D0%BC%D1%96%D1%89%D0%B5%D0%BD%D0%BD%D1%8F

5. Аналіз сучасних послуг оренди житла. URL: https://www.zfort.com.ua/blog/category/article/yak-suchasni-it-tekhnologiyi-zminyuyut-rinok-nerukhomosti

6. Узагальнена інформація про аспекти проєкту. URL: https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/1133

7. Додаткова інформація для створення програмного інтерфейсу URL: https://seo-evolution.com.ua/blog/razrabotka/modernizatsiya-saytu-platform-orendi-neruhomosti

8. Схеми бази даних. URL: https://foxminded.ua/skhemy-bazy-danyh/

9. Нормалізація відношень БД. URL: https://rdb.dp.ua/uk/chapter_03

10. Структура для Backend додатків. URL: https://www.youtube.com/watch?v=AxjsOZCNkJI

11. Піраміда тестування. URL: https://campus.epam.ua/ua/blog/625

ДОДАТОК А ФРАГМЕНТИ ЛІСТИНГУ

```python
from flask import request
from flask_socketio import send, emit

from config import create_app
from db import db
from search import es


app, socketio = create_app()
app.config["INITIALIZED"] = False

db.init_app(app)

@socketio.on('user_id')
def user_id(data):
    print(f"User {data} have been connected")
    print(request.sid)


@socketio.on('sync_message')
def message(data):
    print(data)
    emit("sync_message", data, broadcast=True)


@app.before_request
def init_request():
    if not app.config["INITIALIZED"]:
        db.crea
        try:
            es.reindex_homes()
        except Exception as e:
            print(e)
        app.config["INITIALIZED"] = True


if __name__ == "__main__":
    app.run()
```

(method) def reindex_homes() -> (Any | None)

Рисунок А.1 – Ініціалізація сервісу та WebSocket

Джерело: розроблено автором

```
    def create_index(self):
        self.es.indices.delete(index="real_estate_homes", ignore_unavailable=True)
        self.es.indices.create(index="real_estate_homes", body=mapping)

    def insert_document(self, document):
        return self.es.index(index="real_estate_homes", body=document)

    def insert_documents(self, documents):
        operations = []
        for document in documents:
            operations.append({"index": {"_index": "real_estate_homes"}})
            operations.append(document)
        return self.es.bulk(operations=operations)

    def reindex_homes(self):
        if not self._connected:
            return
        self.create_index()
        homes = HomeManager.select_all_homes()
        for home in homes:
            if home.latitude and home.longitude:
                home.pin = {
                    "location": {
                        "lat": float(home.latitude),
                        "lon": float(home.longitude),
                    }
                }
            else:
                home.pin = None
        resp_schema = HomeResponseSchema()
        resp = resp_schema.dump(homes, many=True)
        return self.insert_documents(resp)

    def search(self, **query_args):
        if self._connected:
            return self.es.search(index="real_estate_homes", **query_args)
        return None
```

Рисунок А.2 – Пошуковий движок

Джерело: розроблено автором

```
import { useContext } from 'react';

import { postHome } from '../api/homeApi';
import { UserContext } from '../context/UserProvider';

const usePostHome = () => {
    const { user } = useContext(UserContext)
    const postHomeAction = async (homeInfo) => {
        let returnData
        try {
            console.log(homeInfo)
            const { error, data } = await postHome(homeInfo, user.id);
            if (error) {
                throw Error()
            }
            console.log(data)
            if (data.id) {
                returnData = data
            } else {
                throw Error()
            }
        } catch (error) {
            console.log(error)
            import('react-toastify').then((module) =>
                module.toast.error('Error during creation of home!', {
                    autoClose: 3000,
                    pauseOnHover: false,
                })
```

Рисунок А.3 – Фільтр-запит

Джерело: розроблено автором