МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА MAPPING СИСТЕМИ ДЛЯ СФЕРИ НЕРУХОМОСТІ»**

Виконав: студент 4 курсу, групи 6.1212-пі-2
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.С. Кожадєй
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Мильцев О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри загальної математики, доцент, к.ф.-м.н. Зіновєєв І.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кожадєю Артему Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка Mapping системи для сфери нерухомості

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та вимоги до системи.
2. Пректування системи.
3. Реалізація та тестування.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	19.12.2025	
2.	Збір вихідних даних.	12.01.2026	
3.	Обробка методичних та теоретичних	02.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	23.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2026	

Студент _____ А.С. Кожадєй
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка Mapping системи для сфери нерухомості»: 53 с., 17 рис., 6 табл., 12 джерел, 2 додатки.

АВТЕНТИФІКАЦІЯ, БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ВІЗУАЛІЗАЦІЯ, ГЕОІНФОРМАЦІЙНА СИСТЕМА, КАРТОГРАФІЯ, НЕРУХОМІСТЬ, ПОЛІГОНАЛЬНИЙ ПОШУК, ПРОСТОРОВИЙ АНАЛІЗ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ФІЛЬТРАЦІЯ.

Об’єкт дослідження – процеси візуалізації, аналізу та прийняття рішень у сфері нерухомості на основі геоінформаційних технологій.

Мета роботи – підвищення швидкості та зручності пошуку об’єктів нерухомості шляхом створення веборієнтованої ГІС із просторовою фільтрацією та інтерактивною картою.

Методи дослідження: аналіз предметної області та аналогів, моделювання БД, проєктування клієнт-серверної архітектури, веброзробка з використанням Python, Flask, Leaflet.js, Shapely, функціональне та інтеграційне тестування.

У роботі розглянуто процес створення геоінформаційної системи для візуалізації та аналізу об’єктів нерухомості. Проведено аналіз сучасних ГІС-рішень, виявлено недоліки та сформульовано вимоги. Запропоновано трирівневу архітектуру: клієнт на HTML, CSS, JavaScript з Leaflet, серверна логіка на Flask, база даних SQLite. Реалізовано автентифікацію з розмежуванням ролей, адміністративний інтерфейс, механізм «обраного» та багатокритеріальну фільтрацію. Робота складається зі вступу, трьох розділів, висновків, переліку посилань і додатків. Перший розділ містить аналіз предметної області та вимог, другий – проєктування системи, третій – реалізацію та тестування. Додатки включають SQL-код БД та лістинг маршрутів Flask.

SUMMARY

Bachelor’s qualifying paper “Development of a Real Estate Mapping System”: 53 pages, 17 figures, 6 tables, 12 references, 2 supplements.

AUTHENTICATION, CARTOGRAPHY, DATABASE, FILTRATION, GEOGRAPHIC INFORMATION SYSTEM, POLYGON SEARCH, REAL ESTATE, SOFTWARE, SPATIAL ANALYSIS, VISUALIZATION, WEB APPLICATION.

Object of study: processes of visualization, analysis and decision-making in the real estate sector based on GIS technologies.

Aim: to increase the speed and convenience of property search by creating a web-based GIS with spatial filtering and an interactive map.

Methods: analysis of the subject area and analogues, database modeling, client-server architecture design, web development using Python, Flask, Leaflet.js, Shapely, functional and integration testing.

The paper presents the development of a GIS for visualization and analysis of real estate objects. Existing GIS solutions are analyzed, their drawbacks identified, and system requirements formulated. A three-tier architecture is proposed: client side on HTML/CSS/JS with Leaflet, server logic on Flask, SQLite database. User authentication with role separation, an administrative interface, a favorites mechanism, and multi-criteria filtering are implemented. The paper consists of an introduction, three chapters, conclusions, references, and appendices. The first chapter covers domain analysis and requirements, the second – system design, the third – implementation and testing. Appendices contain SQL code and Flask route listings.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 7
1 Аналіз предметної області 9
1.1 Аналіз ринку та потреб у геоінформаційних технологіях 9
1.2 Огляд існуючих ГІС-рішень та обґрунтування розробки 10
1.3 Функціональні та нефункціональні вимоги 11
1.4 Вибір архітектури та інструментів реалізації 13
2 Проєктування системи 16
2.1 Проєктування архітектури системи 16
2.2 Проєктування бази даних 17
2.3 Моделювання поведінки системи 20
3 Реалізація та тестування 24
3.1 Особливості реалізації серверної частини 24
3.2 Особливості реалізації клієнтської частини 27
3.3 Інтерфейс користувача 30
3.4 Тестування системи 40
Висновки 43
Перелік посилань 44
Додаток А SQL-код створення таблиць бази даних 45
Додаток Б Основні маршрути Flask-сервера 46

ВСТУП

У сучасному цифровому суспільстві ринок нерухомості переживає активну трансформацію, де ключову роль відіграють геоінформаційні технології. Інтерактивні карти, просторовий аналіз та візуалізація даних стають невід'ємними інструментами для покупців, орендарів, рієлторів та містобудівників. Однак більшість існуючих вебсервісів, таких як OLX.ua та DOM.RIA, пропонують лише базове картографічне відображення без можливості гнучкої просторової фільтрації, зокрема пошуку в довільному полігоні, а також не забезпечують зручного інструментарію для адміністрування об'єктів та персоналізації через «обрані». Отже, актуальною є задача розробки веборієнтованої геоінформаційної системи, яка б поєднувала інтерактивну карту, багатокритеріальну фільтрацію, просторовий аналіз, автентифікацію користувачів і зручний інтерфейс для керування даними.

Об'єктом дослідження є процеси візуалізації, аналізу та прийняття рішень у сфері нерухомості на основі геоінформаційних технологій.

Предмет дослідження – методи та програмні засоби інтерактивної роботи з просторовими даними об'єктів нерухомості, зокрема їх відображення, фільтрації, полігонального пошуку, адміністрування та інтеграції з картографічними сервісами.

Метою роботи є підвищення швидкості та зручності пошуку об'єктів нерухомості для кінцевого користувача шляхом створення веборієнтованої геоінформаційної системи з просторовою фільтрацією та інтерактивною картою.

Для досягнення поставленої мети необхідно вирішити такі основні **завдання**:

– виконати аналіз предметної області та огляд існуючих геоінформаційних систем у сфері нерухомості, виявити їхні переваги та недоліки;

- сформулювати функціональні та нефункціональні вимоги до системи, що розробляється;
- обґрунтувати вибір архітектурного рішення та інструментів реалізації;
- розробити структуру бази даних для зберігання просторових і описових характеристик об’єктів нерухомості, а також інформації про користувачів та їхні вподобання;
- реалізувати клієнтську частину з інтерактивною картою, панеллю фільтрів, списком об’єктів і детальною інформаційною панеллю;
- забезпечити функціонал просторового аналізу та атрибутивної фільтрації;
- створити систему автентифікації користувачів та механізм збереження обраних об’єктів, а також адміністративний інтерфейс для керування об’єктами та їхніми зображеннями;
- провести тестування працездатності розробленої системи та оцінити відповідність заявленим вимогам.

Практичне значення роботи полягає у створенні готового до використання вебзастосунку, який може бути впроваджений в агентствах нерухомості, міських аналітичних відділах або використовуватись як основа для подальшого розширення, зокрема інтеграції з кадастровими даними чи додавання методів машинного навчання для прогнозування цін.

Робота складається зі вступу, трьох розділів, висновків, переліку посилань і двох додатків. У першому розділі проведено аналіз предметної області, сформульовано вимоги до системи та обґрунтовано вибір архітектури й інструментів. У другому розділі виконано проєктування системи, включаючи архітектуру, структуру бази даних та моделювання поведінки. У третьому розділі описано особливості реалізації серверної та клієнтської частин, інтерфейс користувача та результати тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз ринку та потреб у геоінформаційних технологіях

Сучасний ринок нерухомості, зокрема в Україні, перебуває у стані активної цифрової трансформації. Дедалі більша частка угод розпочинається з онлайн-пошуку: потенційні покупці, орендарі та інвестори очікують не лише перегляду текстових оголошень і фотографій, а й можливості оцінити просторове розташування об’єкта, його близькість до транспортних розв’язок, соціальної інфраструктури, зон відпочинку тощо. Саме геоінформаційні технології здатні задовольнити цю потребу, надаючи інтерактивні карти, просторову фільтрацію та візуальний аналіз даних.

Традиційні платформи оголошень часто обмежуються статичними картами з позначками, не пропонуючи користувачеві гнучких інструментів для роботи з простором. У результаті покупець витрачає значний час на рутинне зіставлення адрес із зовнішніми картографічними сервісами, не маючи змоги, наприклад, швидко окреслити на карті бажаний квартал і побачити всі пропозиції в ньому.

Крім того, рієлторські агентства та приватні забудовники зацікавлені у зручних засобах самостійного керування власною базою об’єктів, чого більшість публічних сервісів не передбачають.

Отже, на ринку існує стійкий попит на веборієнтовану ГІС, яка б поєднувала інтерактивну карту, багатокритеріальну атрибутивну фільтрацію, просторовий пошук у довільному полігоні, зручний адміністративний інтерфейс та персоналізовані функції. Розробка такої системи дозволить суттєво підвищити швидкість і зручність пошуку нерухомості для кінцевих споживачів, а також надасть агентствам ефективний інструмент презентації власних пропозицій.

1.2 Огляд існуючих ГІС-рішень та обґрунтування розробки

Сьогодні на світовому ринку представлено низку потужних геоінформаційних платформ, що використовуються у сфері нерухомості. Закордонні лідери, такі як Zillow, пропонують детальні карти з оцінкою вартості житла, історією продажів та прогнозами. Системи рівня ArcGIS Urban та LandVision орієнтовані на містобудівний аналіз та комерційну нерухомість, проте вони є складними, вартісними й надлишковими для повсякденних завдань пересічного покупця чи невеликого агентства.

В Україні найбільш популярними платформами для пошуку нерухомості залишаються DOM.RIA та OLX.ua. Вони надають базове картографічне відображення об’єктів із можливістю фільтрації за типом, ціною, площею та районом. Проте проведений аналіз виявив у них низку принципових недоліків, що обмежують ефективність роботи користувачів.

Відсутність повноцінного просторового аналізу. Жодна з популярних українських платформ не дозволяє виконувати пошук об’єктів у довільному полігоні, накресленому безпосередньо на карті. Користувач змушений обирати із заздалегідь визначених адміністративних районів, що не завжди відповідає його реальним уподобанням щодо конкретного мікрорайону чи кварталу.

Обмежена гнучкість фільтрації. Як правило, фільтри за ціною та площею мають фіксовані діапазони або не дозволяють одночасного комбінування всіх критеріїв із довільними значеннями. Це ускладнює точний підбір об’єктів.

Закритість API. Більшість українських сервісів не надають відкритого програмного інтерфейсу для інтеграції з іншими системами, що унеможливлює побудову на їх основі спеціалізованих аналітичних модулів.

Відсутність інструментів адміністрування для користувачів. Звичайний користувач або представник агентства не може самостійно додати, редагувати чи видалити об’єкт через вебінтерфейс; для цього потрібно

користуватися внутрішніми кабінетами платформ, які часто мають обмежений функціонал або вимагають окремих договорів.

Відсутність зручного механізму збереження обраного. Функція «обране» або відсутня, або реалізована незручно, без можливості швидкого перегляду на карті та переходу до детальної інформації. Зазначені недоліки свідчать про те, що існуючі публічні рішення не задовольняють повною мірою потреби ні кінцевих покупців, ні професійних учасників ринку. Це обґрунтовує доцільність створення власної геоінформаційної системи, вільної від перерахованих обмежень, із відкритою архітектурою, гнучкими засобами просторового пошуку та зручним адміністративним інтерфейсом.

1.3 Функціональні та нефункціональні вимоги

На основі аналізу предметної області та виявлених недоліків аналогів було сформульовано вимоги до розроблюваної ГІС. Вони охоплюють як функціональні можливості, так і якісні характеристики системи. Сформулюємо **функціональні вимоги** до розроблюваної системи.

Візуалізація об’єктів на карті. Система повинна відображати об’єкти нерухомості на інтерактивній карті з використанням бібліотеки Leaflet.js. Мають підтримуватися масштабування, переміщення, кластеризація маркерів для зручного сприйняття великої кількості пропозицій.

Два режими перегляду. Користувачеві надається можливість перемикатися між режимом «Карта» та «Список».

Детальна інформація про об’єкт. При кліку на маркер або картку у списку має відкриватися детальна панель із повним описом: ціна, адреса, район, тип, площа, координати, опис, галерея фотографій із навігацією та можливістю збільшення.

Фільтрація та пошук. Система повинна забезпечувати: фільтрацію за типом нерухомості; фільтрацію за районом; фільтрацію за довільним

діапазоном ціни та площі; просторовий пошук шляхом малювання довільного полігону на карті: після завершення малювання система має показати лише ті об’єкти, координати яких потрапляють у задану область.

Управління користувачами. Будь-який відвідувач може зареєструватися, вказавши ім’я користувача та пароль. Пароль має зберігатися у хешованому вигляді. Зареєстровані користувачі можуть входити в систему, переглядати свій профіль, додавати об’єкти в обране та видаляти їх звідти.

Адміністрування об’єктів. Користувач із роллю адміністратора повинен мати змогу: додавати новий об’єкт нерухомості через форму (назва, опис, ціна, площа, тип, район, адреса, координати). Координати можуть визначатися автоматично через геокодер або вручну кліком на міні-карті у формі; завантажувати для об’єкта декілька фотографій, видаляти їх, змінювати порядок перетягуванням; редагувати будь-який об’єкт (змінювати поля, додавати/видаляти фото); видаляти об’єкти разом з усіма пов’язаними зображеннями.

Адаптивний інтерфейс. На мобільних пристроях панель фільтрів має ховатися за спеціальну кнопку та виїжджати з оверлеєм. Детальна панель об’єкта повинна перебудовуватися так, щоб фотографії знаходилися зверху, а текстова інформація – знизу. Для зручності на мобільних пристроях передбачено кнопку-свайп для виклику фільтрів.

Тепер розглянемо **нефункціональні вимоги.**

Продуктивність. Час завантаження головної сторінки з картою – не більше 3 секунд за нормальних умов мережі. Відгук інтерфейсу на дії користувача – не більше 0,5 секунди. Просторовий пошук у полігоні для вибірки до 1000 об’єктів повинен виконуватися менш ніж за 1 секунду. Серверна частина має витримувати одночасну роботу щонайменше 10 користувачів без істотного зниження швидкодії.

Безпека. Паролі користувачів зберігаються виключно у вигляді bcrypt-хешів. Усі запити до бази даних виконуються через параметризовані SQL-запити для запобігання SQL-ін’єкціям. Виведення даних на клієнті

супроводжується екрануванням HTML-символів для уникнення XSS-атак. Завантажені зображення перевіряються за магічними байтами, дозволені типи JPEG, PNG, GIF та обмежуються розміром 5 МБ. Доступ до адміністративних функцій заборонений для неавторизованих користувачів та користувачів зі звичайною роллю.

Надійність. Система має коректно обробляти виняткові ситуації як на сервері, так і на клієнті, з виведенням зрозумілих повідомлень. Операції додавання/редагування об’єкта та його фотографій виконуються в межах транзакцій для забезпечення цілісності даних.

Зручність використання. Інтерфейс має бути інтуїтивно зрозумілим: основні елементи керування (фільтри, перемикач «Карта/Список», кнопка обраного) знаходяться на видних місцях, форми супроводжуються підказками. Адаптивний дизайн забезпечує комфортну роботу як із десктопів, так і з мобільних пристроїв.

Масштабованість. Архітектура системи повинна передбачати можливість заміни SQLite на потужнішу СКБД без значних змін у кодовій базі, а також додавання нових функціональних модулів.

1.4 Вибір архітектури та інструментів реалізації

Для реалізації системи, що відповідає сформульованим вимогам, було обрано трирівневу клієнт-серверну архітектуру, яка забезпечує чітке розділення обов’язків, спрощує супровід та масштабування. Презентаційний рівень реалізований як односторінковий вебзастосунок на основі HTML5, CSS3 та JavaScript. Уся взаємодія з сервером відбувається асинхронно, що забезпечує плавність інтерфейсу без перезавантаження сторінки.

Рівень бізнес-логіки побудований на мікрофреймворку Flask [1]. Вибір Flask зумовлений його легкістю, гнучкістю та багатою екосистемою розширень.

Flask-Login використовується для керування сесіями користувачів та захисту маршрутів, що потребують автентифікації [5]. Flask-Bcrypt забезпечує надійне хешування паролів [6].

Shapely – бібліотека для роботи з геометричними об’єктами, яка застосовується для виконання просторових запитів (перевірка входження точки в полігон) [4].

Werkzeug використовується для безпечної обробки імен файлів при завантаженні зображень. Рівень збереження даних реалізований за допомогою реляційної бази даних SQLite [3]. Такий вибір обґрунтований простотою розгортання, легкістю резервного копіювання та достатньою продуктивністю для очікуваних обсягів даних. Детальне проєктування структури бази даних винесено в наступний розділ роботи, проте вже на етапі вибору архітектури було враховано можливість подальшої міграції на PostgreSQL із розширенням PostGIS при зростанні навантаження.

Для побудови інтерактивної карти обрано бібліотеку Leaflet.js – легке, відкрите рішення, яке підтримує всі необхідні функції: роботу з тайлами OpenStreetMap, кластеризацію маркерів, малювання полігонів та адаптивне відображення [2]. Важливим фактором є відсутність залежності від пропрієтарних API, що робить систему незалежною та придатною для вільного використання.

Для автоматичного визначення координат за адресою інтегровано два зовнішні сервіси: PositionStack та Nominatim [8]. Передбачено також можливість ручного уточнення координат адміністратором безпосередньо на карті у формі додавання/редагування об’єкта, що компенсує можливі похибки безкоштовних геокодерів.

Безпека файлових операцій забезпечується перевіркою магічних байтів для визначення справжнього типу зображення JPEG, PNG, GIF, а не лише розширення, що унеможливлює завантаження шкідливих файлів під виглядом зображень. Унікальні імена файлів генеруються з додаванням випадкового суфікса для запобігання колізіям.

Таким чином, обрані архітектурне рішення та стек технологій дозволяють створити гнучку, безпечну та продуктивну геоінформаційну систему, яка повністю відповідає поставленим вимогам і може бути реалізована в межах кваліфікаційної роботи.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Проєктування архітектури системи

На основі обраної трирівневої клієнт-серверної моделі було розроблено архітектуру геоінформаційної системи, яка забезпечує чітке розмежування відповідальності між компонентами, спрощує тестування, супровід та масштабування. Архітектурна схема системи представлена на рисунку 2.1.

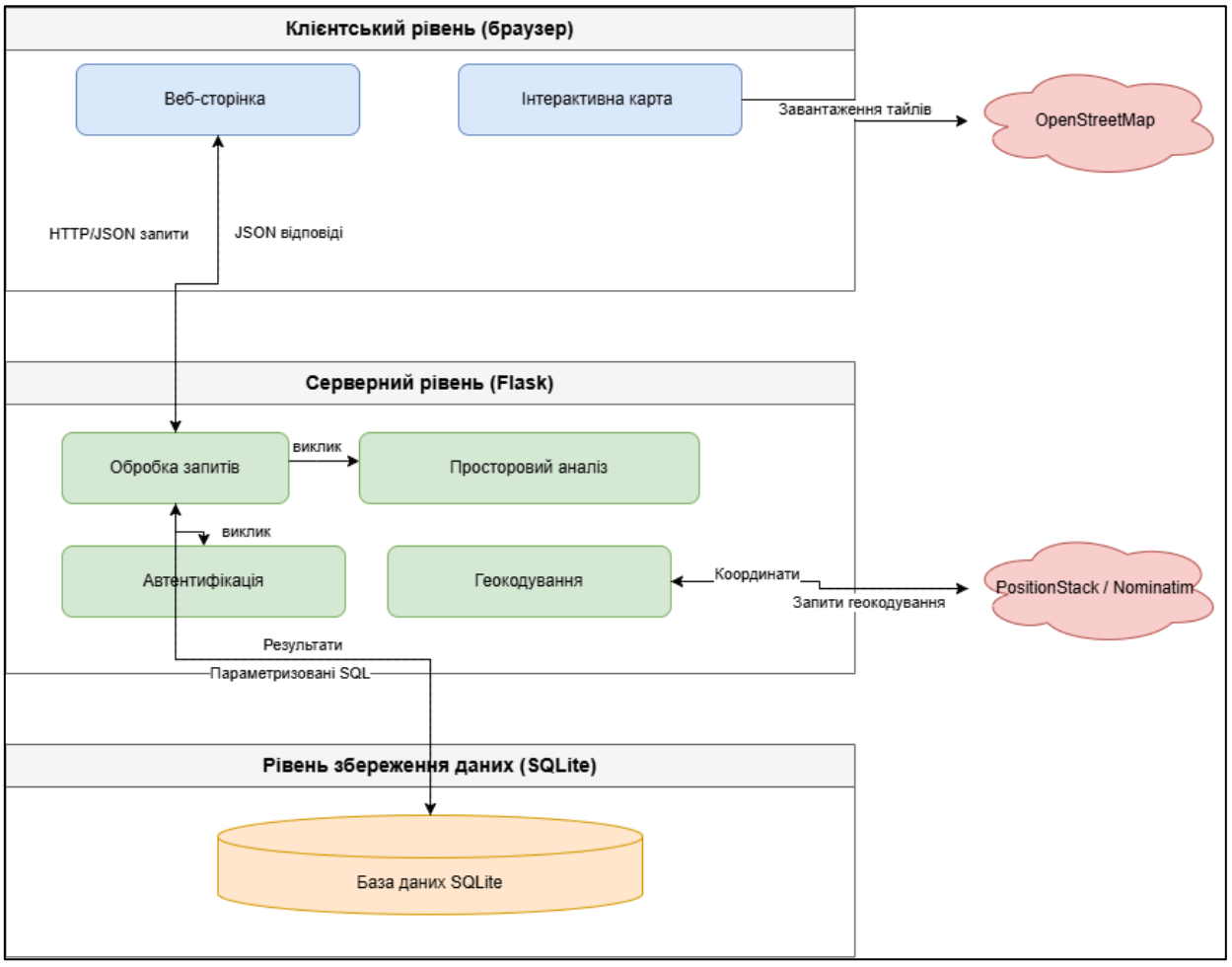


Рисунок 2.1 – Архітектура ГІС для пошуку об’єктів нерухомості

Перший рівень – *рівень представлення* – реалізований у вигляді односторінкового вебзастосунку, що виконується в браузері користувача. Він включає HTML-структуру, каскадні таблиці стилів для адаптивного дизайну

та JavaScript-логіку, яка відповідає за інтерактивність: відображення карти Leaflet з маркерами, обробку подій, асинхронну взаємодію з сервером через AJAX-запити у форматі JSON.

Другий рівень – *рівень бізнес-логіки* – розгорнуто на вебсервері Flask. Він приймає HTTP-запити від клієнта, виконує автентифікацію та авторизацію за допомогою розширень Flask-Login та Flask-Bcrypt, обробляє запити на фільтрацію, сортування, додавання/редагування/видалення об’єктів, а також реалізує просторовий аналіз: для полігонального пошуку сервер отримує координати вершин полігону, за допомогою бібліотеки Shapely створює об’єкт Polygon, після чого перевіряє входження координат кожного об’єкта нерухомості в цей полігон. Усі відповіді сервера повертаються у вигляді JSON-документів, що робить API універсальним для потенційних мобільних клієнтів.

Третій рівень – *рівень збереження даних* – представлений реляційною базою даних SQLite. Доступ до бази здійснюється через параметризовані SQL-запити, що гарантує захист від SQL-ін’єкцій. Для пришвидшення операцій читання створено індекси на полях, які найчастіше фігурують у фільтрах: тип нерухомості, район, ціна. Додатково система інтегрується із зовнішніми геокодерами для перетворення адреси в координати, що використовується при додаванні нових об’єктів адміністратором. Така архітектура забезпечує низьку зв’язність між рівнями: заміна бази даних на PostgreSQL/PostGIS або модифікація клієнтської частини не потребуватимуть переписування всієї системи.

2.2 Проєктування бази даних

Проєктування структури даних виконано у два етапи: концептуальне моделювання, що відображає основні сутності предметної області та зв’язки між ними у вигляді ER-діаграми, та логічне моделювання, яке деталізує ці

сутності у формі реляційних таблиць з визначенням типів даних та обмежень цілісності.

Концептуальна ER-модель системи наведена на рисунку 2.2. У нотації «вороняча лапка» на ній зображено п’ять основних сутностей: Property, PropertyImage, User, Favorite та SearchLog. Сутність Property є центральною: вона містить атрибути, що описують об’єкт – унікальний ідентифікатор, назву, опис, ціну, площу, тип, район, адресу, широту та довготу. Зв’язок між Property та PropertyImage має тип «один-до-багатьох»: один об’єкт може мати багато зображень, кожне зображення належить лише одному об’єкту.

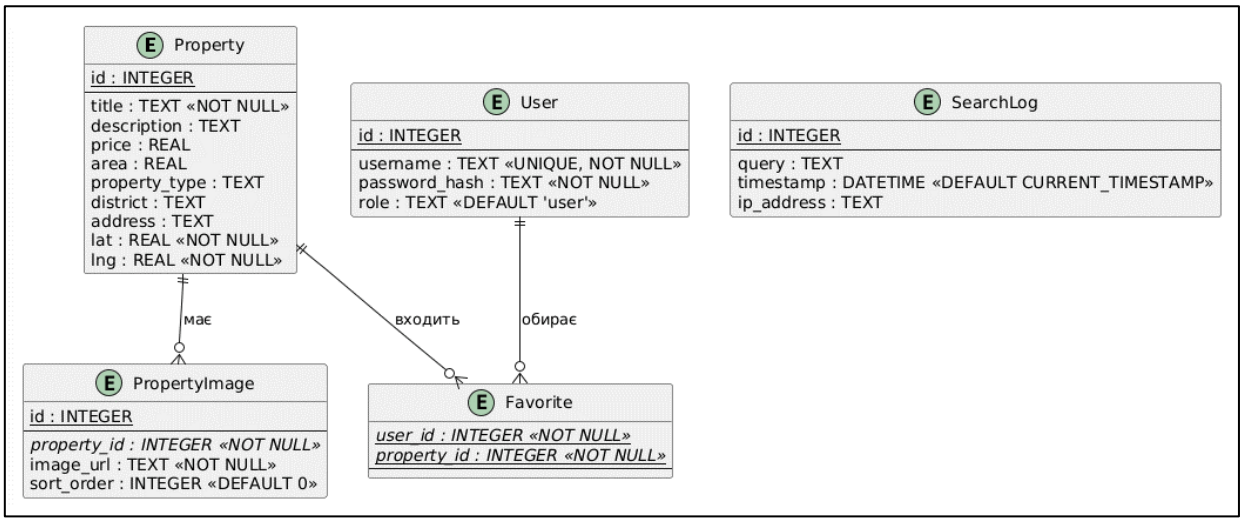


Рисунок 2.2 – ER-діаграма бази даних

Для підтримки довільного порядку фотографій у сутності PropertyImage передбачено атрибут sort_order. Сутність User зберігає облікові дані: логін, хеш пароля та роль (адміністратор або звичайний користувач). Зв’язок «багато-до-багатьох» між User та Property реалізовано через асоціативну сутність Favorite, яка має складовий первинний ключ із зовнішніх ключів на обидві таблиці. Це дозволяє кожному користувачеві формувати власний список обраних об’єктів.

Сутність SearchLog фіксує пошукові запити, час та IP-адресу для майбутньої аналітики, хоча в поточній версії цей функціонал ще не активовано на клієнтському рівні.

Логічна модель бази даних, отримана шляхом відображення ER-діаграми на реляційну схему, представлена в таблицях 2.1 – 2.5. Для кожної таблиці наведено ім’я атрибута, його тип даних та обмеження цілісності.

Таблиця 2.1 – Структура таблиці Property

Атрибут	**Тип**	**Обмеження**
id	INTEGER	PRIMARY KEY, AUTOINCREMENT
title	TEXT	NOT NULL
description	TEXT	
price	REAL	
area	REAL	
property_type	TEXT	
district	TEXT	
address	TEXT	
lat	REAL	NOT NULL
lng	REAL	NOT NULL

Таблиця 2.2 – Структура таблиці User

Атрибут	**Тип**	**Обмеження**
id	INTEGER	PRIMARY KEY, AUTOINCREMENT
username	TEXT	UNIQUE, NOT NULL
password_hash	TEXT	NOT NULL
role	TEXT	DEFAULT 'user'

Таблиця 2.3 – Структура таблиці Favorite

Атрибут	**Тип**	**Обмеження**
user_id	INTEGER	NOT NULL, FOREIGN KEY REFERENCES User(id)
property_id	INTEGER	NOT NULL, FOREIGN KEY REFERENCES Property(id)
PRIMARY KEY (user_id, property_id)		

Таблиця 2.4 – Структура таблиці SearchLog

Атрибут	**Тип**	**Обмеження**
id	INTEGER	PRIMARY KEY, AUTOINCREMENT
query	TEXT	

Продовження таблиці 2.4

Атрибут	Тип	Обмеження
timestamp	TEXT	DEFAULT CURRENT_TIMESTAMP
ip_address	TEXT	

Таблиця 2.5 – Структура таблиці PropertyImage

Атрибут	Тип	Обмеження
id	INTEGER	PRIMARY KEY, AUTOINCREMENT
property_id	INTEGER	NOT NULL, FOREIGN KEY REFERENCES
Property(id) ON DELETE CASCADE		
image_url	TEXT	NOT NULL
sort_order	INTEGER	DEFAULT 0

Усі таблиці нормалізовано до третьої нормальної форми, що виключає аномалії оновлення та забезпечує цілісність даних. Зовнішні ключі з опцією ON DELETE CASCADE гарантують автоматичне видалення пов’язаних зображень при видаленні об’єкта нерухомості. Повний SQL-код створення таблиць винесено в Додаток А.

2.3 Моделювання поведінки системи

Для специфікації функціональних можливостей системи та сценаріїв взаємодії користувачів із нею було розроблено діаграми мовою UML. На рисунку 2.3 зображено діаграму прецедентів, яка окреслює межі системи, її акторів та основні варіанти використання. Акторами виступають «Неавторизований користувач», «Авторизований користувач» та «Адміністратор».

Неавторизований користувач може лише переглядати об’єкти на карті та в списку, застосовувати атрибутивні фільтри та малювати полігон для просторового пошуку.

Авторизований користувач, окрім цього, отримує змогу реєструватися, входити в систему, додавати об’єкти до обраного та переглядати їх список, а також виходити з облікового запису.

Адміністратор успадковує всі можливості авторизованого користувача, але додатково йому доступні прецеденти керування об’єктами: додавання нового об’єкта (з геокодуванням адреси або ручним вибором координат), завантаження фотографій, редагування атрибутів та зображень, зміна порядку фото, видалення об’єкта та його зображень. Зв’язок узагальнення між акторами підкреслює ієрархію прав.

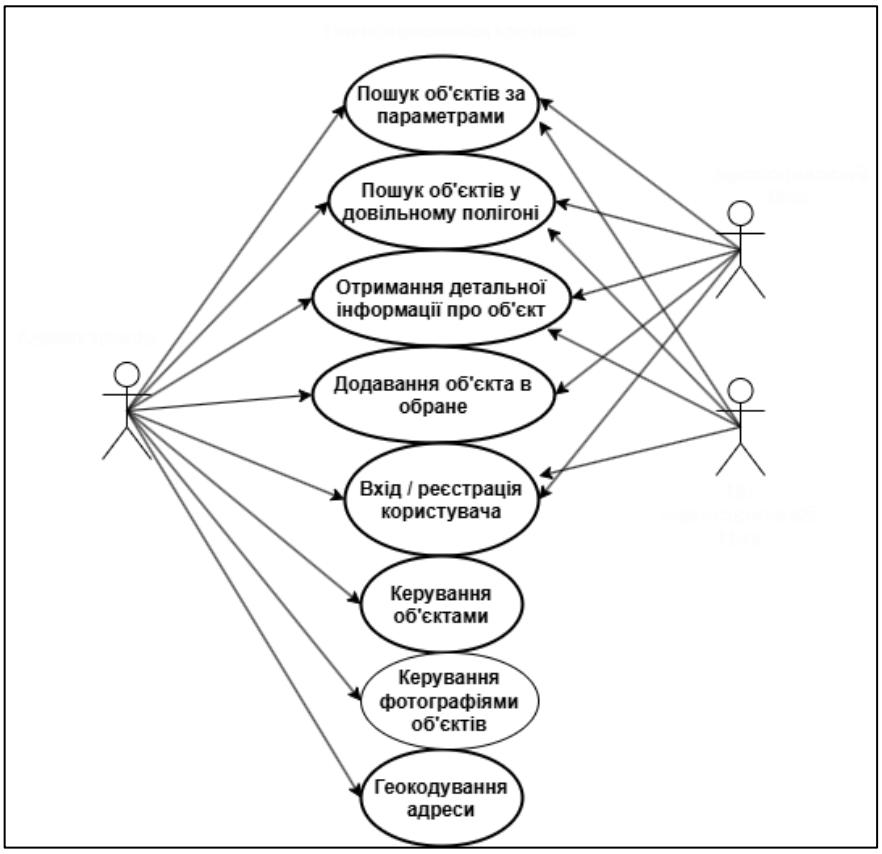


Рисунок 2.3 – Діаграма прецедентів

Для деталізації одного з ключових прецедентів – «Пошук об’єктів у довільному полігоні» – побудовано діаграму діяльності, подану на рисунку 2.4. Вона описує бізнес-процес з точки зору користувача. Початковим вузлом є бажання користувача знайти нерухомість у конкретній просторовій зоні.

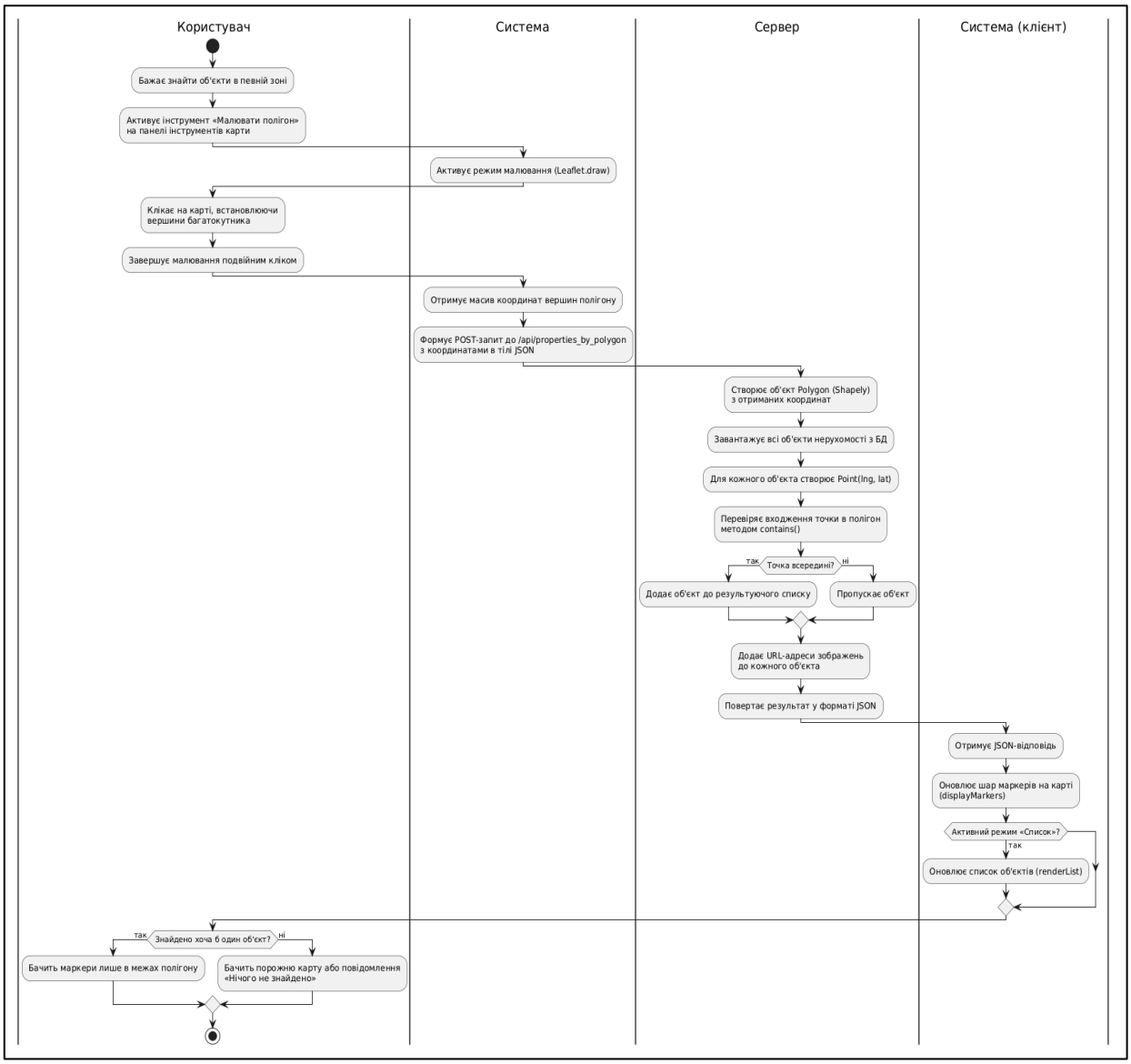


Рисунок 2.4 – Діаграма діяльності прецеденту
«Пошук об’єктів у довільному полігоні»

Користувач активує інструмент малювання полігону на панелі інструментів карти. Далі він послідовно клацає на карті, встановлюючи вершини багатокутника, і завершує малювання подвійним кліком. Система отримує масив координат вершин і надсилає асинхронний POST-запит на серверний маршрут /api/properties_by_polygon.

Сервер, використовуючи бібліотеку Shapely, створює об’єкт Polygon та завантажує всі об’єкти нерухомості з бази даних. Для кожного об’єкта формується точка Point за його координатами, після чого перевіряється входження точки в полігон методом contains. Об’єкти, що пройшли перевірку,

збагачуються URL-адресами зображень та повертаються клієнту у форматі JSON.

Клієнтський скрипт оновлює шар маркерів на карті та, за потреби, список об’єктів, відображаючи лише ті, що потрапили в накреслену область. Якщо жоден об’єкт не задовольняє умову, користувач бачить порожню карту або відповідне повідомлення. Діаграма наочно демонструє розгалуження логіки та взаємодію між рівнями системи, не заглиблюючись у технічні деталі реалізації, які розкрито в наступному розділі.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Особливості реалізації серверної частини

Серверна частина геоінформаційної системи реалізована на мові Python із застосуванням мікрофреймворку Flask. Основними завданнями серверної логіки є обслуговування REST API, взаємодія з базою даних SQLite, виконання просторових запитів, забезпечення автентифікації та авторизації користувачів, а також безпечне збереження завантажених зображень.

Усі маршрути, що повертають дані про об’єкти нерухомості, підтримують багатокритеріальну фільтрацію та сортування. Запити на отримання переліку об’єктів обробляються з урахуванням параметрів рядка запиту: тип нерухомості, район, мінімальна та максимальна ціна, мінімальна та максимальна площа. Для запобігання SQL-ін’єкціям усі значення фільтрів передаються в базу даних виключно через параметризовані запити.

Фільтрація реалізована через динамічне складання SQL-запиту в маршруті /api/properties. Базовий запит SELECT * FROM property WHERE 1=1 доповнюється умовами лише за наявності відповідних параметрів. Для кожного параметра додається окрема умова з плейсхолдером, а значення додається до списку параметрів. Такий підхід забезпечує, по-перше, абсолютний захист від SQL-ін’єкцій (жодне значення не вставляється в запит напряму), а по-друге, гнучкість – користувач може комбінувати будь-який набір фільтрів, і сервер згенерує оптимальний запит. Після отримання результатів з бази даних, якщо користувач обрав сортування за ціною, вибірка сортується на стороні сервера за допомогою функції sorted із використанням безпечного перетворення ціни через допоміжну функцію safe_price_value.

Функція safe_price_value відповідає за безпечну обробку числових значень, що надходять від користувача. Оскільки поля фільтрації (ціна, площа) можуть містити невалідні символи або передаватися у некоректному форматі, вона виконує очищення вхідного рядка від усіх символів, окрім цифр, крапки

та мінуса, замінює кому на крапку (що є типовою помилкою при ручному введенні) та безпечно перетворює результат у число з плаваючою комою. У разі неможливості перетворення функція повертає нульове значення, що запобігає виникненню винятків під час виконання запиту.

Ключовим елементом серверної логіки, що вирізняє розроблену систему серед аналогів, є реалізація просторового пошуку об’єктів у довільному полігоні. Алгоритм працює наступним чином: клієнт надсилає POST-запит із масивом координат вершин полігону, накресленого користувачем. Сервер за допомогою бібліотеки Shapely створює об’єкт Polygon, після чого витягує всі записи з таблиці Property. Для кожного запису формується точка Point з координатами lng та lat. Метод contains бібліотеки Shapely перевіряє входження точки в полігон, і об’єкти, що задовольняють умову, збагачуються URL-адресами зображень і повертаються у форматі JSON. Рисунок 3.1 демонструє реалізацію цього алгоритму.

```
@app.route('/api/properties_by_polygon', methods=['POST'])
def properties_by_polygon():
    data = request.get_json()
    if not data or 'polygon' not in data:
        return jsonify([])
    coords = data['polygon']
    try:
        poly = Polygon(coords)
    except:
        return jsonify({'error': 'Некоректний полігон'}), 400

    conn = get_db()
    c = conn.cursor()
    c.execute("SELECT * FROM property")
    rows = c.fetchall()
    result = []
    for r in rows:
        d = dict(r)
        c2 = conn.cursor()
        c2.execute("SELECT image_url FROM property_images WHERE property_id=? ORDER BY sort_order",
(d['id'],))
        d['images'] = [row['image_url'] for row in c2.fetchall()]
        c2.close()
        if poly.contains(Point(r['lng'], r['lat'])):
            result.append(d)
    conn.close()
    return jsonify(result)
```

Рисунок 3.1 – Обробник полігонального пошуку

Обраний підхід є оптимальним для локальних ринків із кількістю об’єктів до кількох тисяч, оскільки не потребує просторових розширень бази даних і забезпечує час відповіді менше однієї секунди, що відповідає сформульованим нефункціональним вимогам.

Автентифікація користувачів побудована на зв’язці Flask-Login та Flask-Bcrypt. Паролі ніколи не зберігаються у відкритому вигляді – лише у формі bcrypt-хешів із сіллю. Розмежування прав доступу реалізовано через декоратори: @login_required захищає маршрути, доступні авторизованим користувачам, а спеціально створений декоратор @admin_required додатково перевіряє наявність ролі 'admin' у поточного користувача.

Безпека завантаження зображень забезпечується на декількох рівнях. По-перше, розмір файлу обмежується 5 МБ на рівні конфігурації Flask. По-друге, функція allowed_image виконує перевірку не лише за розширенням файлу, а й за його магічними байтами (сигнатурами). Для JPEG перевіряються байти \xFF\xD8\xFF, для PNG – \x89PNG, для GIF – GIF87a або GIF89a. Така перевірка унеможливлює завантаження виконуваних файлів або скриптів, перейменованих на зображення. По-третє, для кожного завантаженого файлу генерується унікальне ім’я з використанням функції secure_filename (яка видаляє небезпечні символи) та додаванням випадкового шістнадцяткового суфікса, отриманого через os.urandom(4).hex(). Це запобігає як колізіям імен, так і спробам перезаписати існуючі файли.

Для підвищення продуктивності при старті сервера викликається функція ensure_indexes, яка створює індекси на полях property_type, district та price таблиці property, якщо вони ще не існують. Такий підхід гарантує, що навіть при розгортанні системи з нуля база даних одразу буде оптимізована для виконання запитів фільтрації, які є найчастішими в роботі системи.

Система також передбачає ведення логу пошукових запитів через маршрут /api/search_log, який зберігає текст запиту, часову мітку та IP-адресу користувача. Хоча в поточній версії фронтенд не використовує цей

функціонал, він закладений для майбутньої аналітики популярних напрямків пошуку. Видалення об’єктів реалізовано з каскадним видаленням пов’язаних зображень як із бази даних, так і з файлової системи, що підтримує чистоту сховища.

3.2 Особливості реалізації клієнтської частини

Клієнтська частина системи є односторінковим вебзастосунком, інтерфейс якого побудований на основі HTML5, CSS3 та нативного JavaScript. Уся взаємодія з сервером відбувається асинхронно через Fetch API, що дозволяє оновлювати дані на карті та в списку без перезавантаження сторінки.

Центральним елементом візуалізації є інтерактивна карта, реалізована за допомогою бібліотеки Leaflet.js із підключенням плагінів Leaflet.markercluster для кластеризації маркерів [10] та Leaflet.draw для малювання полігонів [11]. Рисунок 3.2 ілюструє ключовий фрагмент клієнтського коду, що відповідає за ініціалізацію карти, налаштування кластеризації та обробку події завершення малювання полігону.

Кластеризація налаштована таким чином, що при наближенні до рівня масштабу 16 і вище кластери автоматично розпадаються на окремі маркери. Функція iconCreateFunction генерує стилізоване коло з кількістю об’єктів у кластері, що покращує сприйняття щільності пропозицій на карті. Інструмент малювання полігону дозволяє створювати довільні багатокутники та прямокутники; після завершення малювання спрацьовує обробник події L.Draw.Event.CREATED, який збирає координати вершин, надсилає їх на сервер і оновлює карту та список на основі отриманої відповіді.

Клієнтська логіка побудована навколо центрального масиву стану allProperties, який зберігає всі поточні об’єкти після застосування фільтрів. Усі операції (перемикання режимів перегляду, додавання в обране, видалення) працюють саме з цим масивом, що забезпечує синхронізацію між

картою та списком без додаткових запитів до сервера. Після кожної зміни даних викликаються функції displayMarkers для оновлення шару маркерів на карті та, за потреби, renderList для оновлення списку. Така архітектура гарантує, що користувач завжди бачить актуальні дані незалежно від активного режиму.

```
map = L.map('map').setView([47.8388, 35.1396], 13);
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
   attribution: '© OpenStreetMap'
}).addTo(map);

markersCluster = L.markerClusterGroup({
   showCoverageOnHover: false,
   maxClusterRadius: 45,
   spiderfyOnMaxZoom: true,
   disableClusteringAtZoom: 16,
   animate: true,
   iconCreateFunction: function(cluster) {
      const count = cluster.getChildCount();
      return L.divIcon({
         html: `<div class="cluster-icon">${count}</div>`,
         className: 'marker-cluster',
         iconSize: L.point(40, 40)
      });
   }
});
map.addLayer(markersCluster);

drawnItems = new L.FeatureGroup();
map.addLayer(drawnItems);
const drawControl = new L.Control.Draw({
   edit: { featureGroup: drawnItems },
   draw: { polygon: true, rectangle: true, circle: false, marker: false, polyline: false }
});
map.addControl(drawControl);

map.on(L.Draw.Event.CREATED, async (e) => {
   drawnItems.clearLayers();
   drawnItems.addLayer(e.layer);
   const coords = e.layer.getLatLngs()[0].map(p => [p.lng, p.lat]);
   const resp = await fetch('/api/properties_by_polygon', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ polygon: coords })
   });
   const data = await resp.json();
   allProperties = data;
   displayMarkers(data);
   if (currentView === 'list') renderList(data);
});
```

Рисунок 3.2 – Ініціалізація карти та обробка малювання полігону

Особливістю реалізації роботи з обраними об’єктами є оптимістичне оновлення інтерфейсу. При натисканні кнопки «В обране» зміна стану кнопки та оновлення відповідних елементів (маркерів, карток, детальної панелі) відбуваються миттєво, не очікуючи відповіді від сервера. Для цього клієнт підтримує локальну множину favoriteIds, яка оновлюється синхронно з дією користувача. Паралельно надсилається асинхронний POST-запит до сервера. У разі помилки мережі стан інтерфейсу відкочується назад, що зберігає узгодженість даних. Для запобігання стану гонитви при швидких повторних кліках використовується блокування через змінну favoriteRequestLock, яка ігнорує повторні запити до завершення поточного.

Функція createPopup відповідає за генерацію HTML-вмісту спливаючих вікон маркерів. Вона динамічно формує вміст залежно від ролі поточного користувача: для адміністратора додаються кнопки редагування та видалення, для звичайного користувача – лише кнопка додавання в обране. Стан кнопки «В обране» визначається на основі локальної множини favoriteIds, що забезпечує миттєве відображення актуального стану без додаткових запитів. Уся вбудована в попап інформація, отримана від користувача або з бази даних, проходить через функцію escapeHtml, яка замінює символи &, <, > на відповідні HTML-сутності, що унеможливлює виконання шкідливого коду через XSS-атаки.

Для збереження стану інтерфейсу між сеансами роботи використовуються механізми localStorage (запам’ятовування обраного режиму «Карта» або «Список») та URL-хешу (для фіксації відкритого об’єкта в детальній панелі). Такий підхід забезпечує зручність користувача при оновленні сторінки або поверненні до неї пізніше.

Адаптивність інтерфейсу реалізована через медіа-запити CSS із контрольним значенням ширини екрана 768 пікселів. На мобільних пристроях панель фільтрів приховується та викликається спеціальною кнопкою-свайпом, а детальна панель об’єкта змінює компонування з двоколонкового на вертикальне, розміщуючи галерею фотографій над текстовим описом.

3.3 Інтерфейс користувача

Інтерфейс розробленої геоінформаційної системи реалізує всі сценарії взаємодії, передбачені функціональними вимогами. Нижче наведено основні екрани системи, що демонструють її роботу в різних режимах.

Головний екран застосунку в десктопній версії представлено на рисунку 3.3. Він має класичне двопанельне компонування: ліворуч розташована панель фільтрів, праворуч – основна робоча область, яка може відображати або інтерактивну карту, або список об’єктів нерухомості. Перемикання між режимами «Карта» та «Список» здійснюється за допомогою двох кнопок, розташованих безпосередньо над робочою областю. Обраний режим зберігається у локальному сховищі браузера, тому після перезавантаження сторінки або повторного відвідування системи користувач автоматично повертається до звичного вигляду.

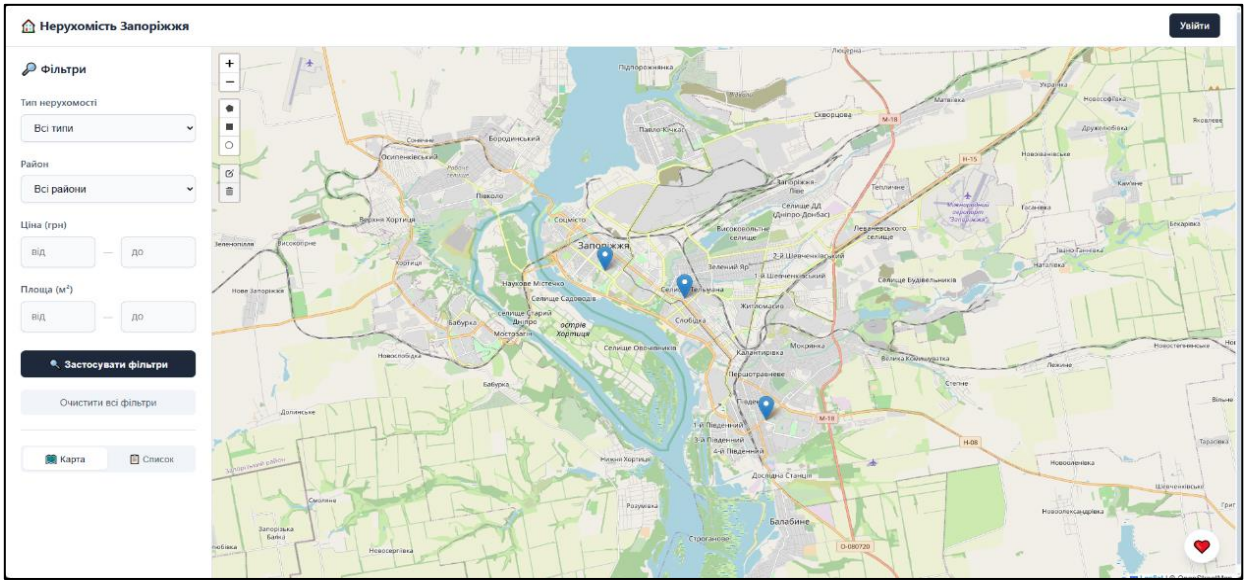


Рисунок 3.3 – Головний екран застосунку

У верхній частині сторінки знаходиться навігаційне меню, що містить назву системи, кнопки «Увійти» для неавторизованих відвідувачів, або ім’я користувача та кнопку «Вийти» для авторизованих. Для користувача з роллю адміністратора в цьому ж меню з’являється додаткова кнопка «Додати об’єкт».

У правому нижньому куті екрана постійно видимою є плаваюча кругла кнопка із зірочкою, яка відкриває модальне вікно зі списком обраних об’єктів. Карта реалізована на базі бібліотеки Leaflet.js із використанням вільних тайлів OpenStreetMap [7].

Початковий центр карти встановлено на місто Запоріжжя з рівнем масштабування 13, що дозволяє охопити основні райони міста. Користувач може вільно переміщувати карту, змінювати масштаб за допомогою коліщатка миші або кнопок навігації.

Кожен об’єкт нерухомості відображається на карті у вигляді маркера. При великій щільності об’єктів автоматично вмикається кластеризація, реалізована через плагін Leaflet.markercluster: кілька сусідніх маркерів об’єднуються в один кластер із зазначенням їхньої кількості.

Кластерне коло має спеціальне стилізоване оформлення, яке змінює відтінок залежно від кількості об’єктів. При досягненні максимального рівня масштабування кластери автоматично розпадаються на окремі маркери, що дозволяє точно визначити розташування кожного об’єкта.

При кліку на маркер відкривається спливаюче вікно (popup), яке містить стислу інформацію: назву об’єкта, його тип, ціну та, за наявності, мініатюру першого зображення. У цьому ж вікні розташовані кнопки швидких дій: «В обране» (для авторизованих користувачів), «Редагувати» та «Видалити» (для адміністратора), а також посилання для відкриття детальної панелі.

Повторний клік на маркер або клік поза спливаючим вікном закриває його.

Панель фільтрів (див. рис. 3.4) надає засоби для багатокритеріального відбору об’єктів за атрибутами. Вона містить два випадаючі списки: «Тип нерухомості» (квартира, будинок, офіс) та «Район». Список районів формується динамічно на основі наявних у базі даних, що гарантує його актуальність. Нижче розташовані чотири числових поля для завдання довільних діапазонів ціни та площі: «Ціна від», «Ціна до», «Площа від», «Площа до».

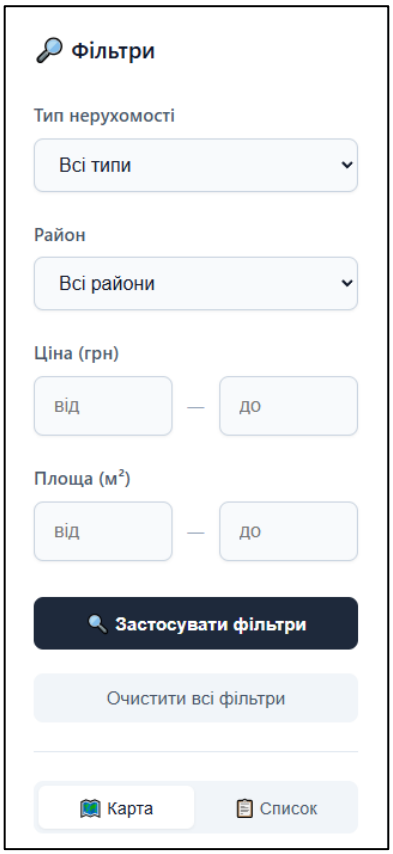


Рисунок 3.4 – Панель фільтрів

Кнопка «Застосувати фільтри» надсилає асинхронний GET-запит до сервера з вибраними параметрами. Після отримання відповіді карта та список миттєво оновлюються, відображаючи лише ті об’єкти, що задовольняють критеріям. Кнопка «Очистити всі фільтри» скидає всі поля до початкових значень і завантажує повний перелік об’єктів. Додатково над списком об’єктів доступне сортування за ціною (за зростанням або спаданням), яке виконується без перезавантаження сторінки.

Унікальною функціональною особливістю системи є можливість пошуку об’єктів у довільному багатокутнику, намальованому користувачем безпосередньо на карті. Для активації цього режиму на панелі інструментів карти передбачено кнопку малювання полігону (плагін Leaflet.draw). Після її натискання користувач послідовними кліками встановлює вершини багатокутника, а подвійний клік завершує малювання. Намальована фігура підсвічується на карті, а раніше накреслені полігони автоматично видаляються, щоб уникнути накладання.

Одразу після завершення малювання клієнтський скрипт збирає координати вершин, формує POST-запит до серверного маршруту /api/properties_by_polygon і передає масив координат у форматі JSON. Сервер, використовуючи бібліотеку Shapely, створює об’єкт Polygon і перевіряє входження кожної точки нерухомості в задану область.

Результатом є список об’єктів, які потрапили в полігон, що повертається клієнту. На карті залишаються лише маркери, що відповідають умові, а список (якщо активний) також фільтрується. Якщо жоден об’єкт не потрапив у вибрану зону, карта стає порожньою, а користувач бачить відповідне повідомлення. Приклад результату полігонального пошуку наведено на рисунку 3.5.

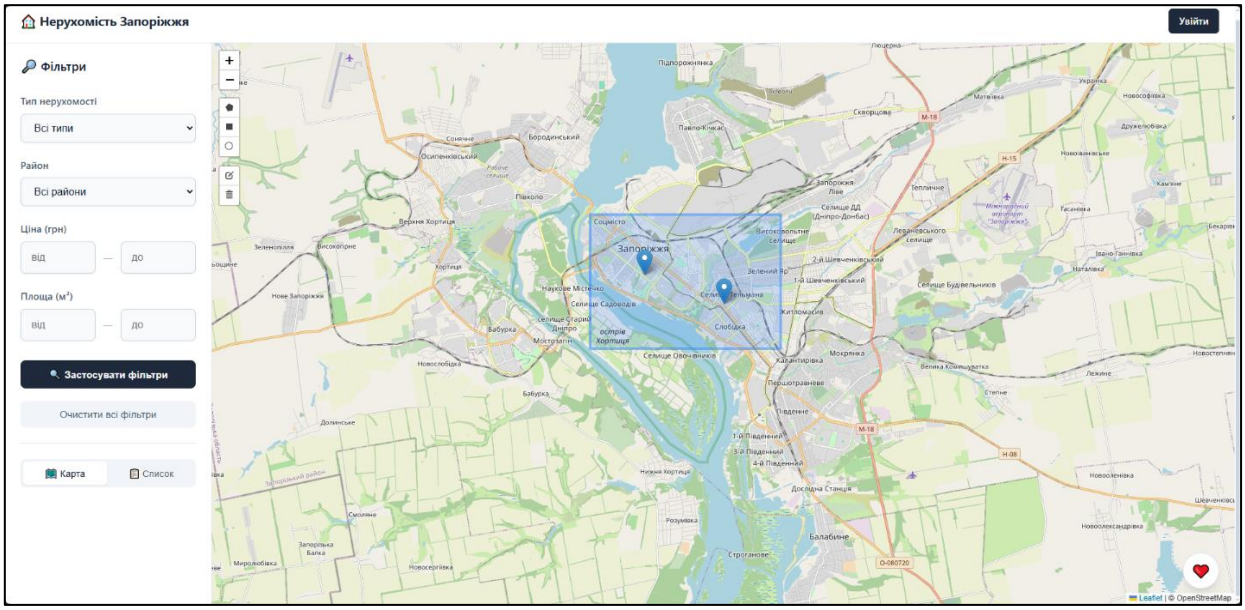


Рисунок 3.5 – Результат полігонального пошуку

Альтернативою картографічному відображенню є табличний режим «Список» (див. рис. 3.6). У цьому режимі об’єкти виводяться у вигляді карток, кожна з яких містить: назву об’єкта, адресу, тип нерухомості, площу та ціну, виділену великим шрифтом. Ліворуч від текстової інформації, за наявності, відображається мініатюра першого зображення об’єкта. Картки мають візуальний зворотний зв’язок: при наведенні курсора вони плавно піднімаються з легкою тінню, що покращує сприйняття інтерактивності.

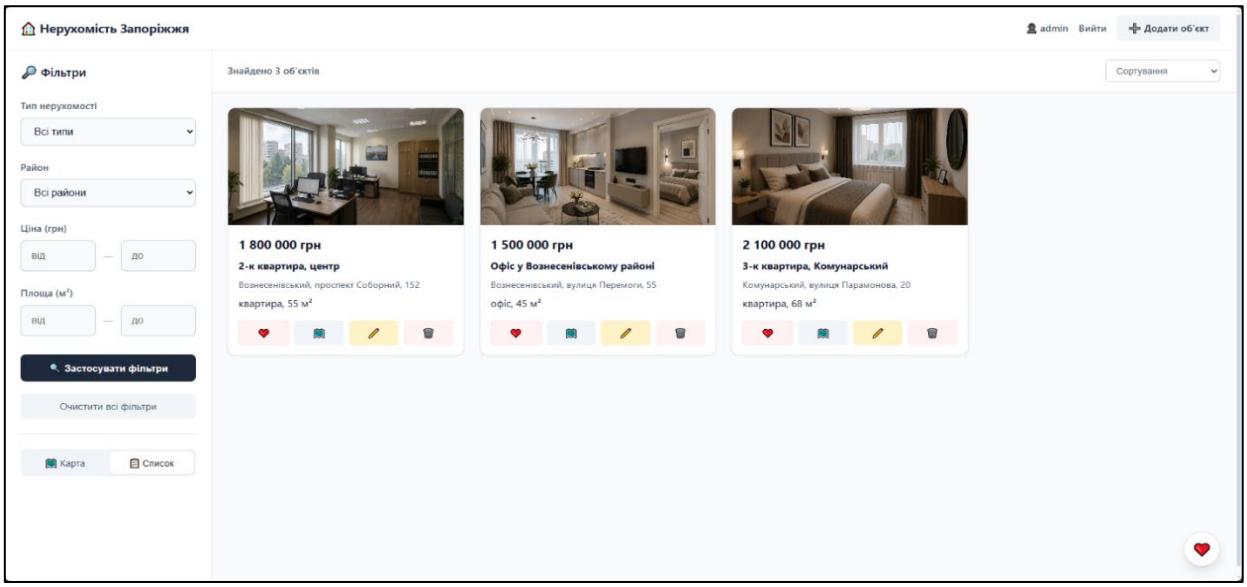


Рисунок 3.6 – Режим «Список»

Кожна картка містить набір кнопок дій: «На карту» перемикає режим перегляду на карту і центрує її на обраному об’єкті, «В обране» для авторизованих) а також «Редагувати» та «Видалити» для адміністратора. Клік на будь-яку область картки відкриває детальну інформаційну панель для цього об’єкта. Сортування списку за ціною доступне через випадаюче меню над списком і виконується без перезавантаження сторінки.

Детальна панель об’єкта (див. рис. 3.7) відкривається при кліку на маркер, на картку в списку або через посилання з модального вікна «Обране». Вона займає всю робочу область, тимчасово перекриваючи карту або список.

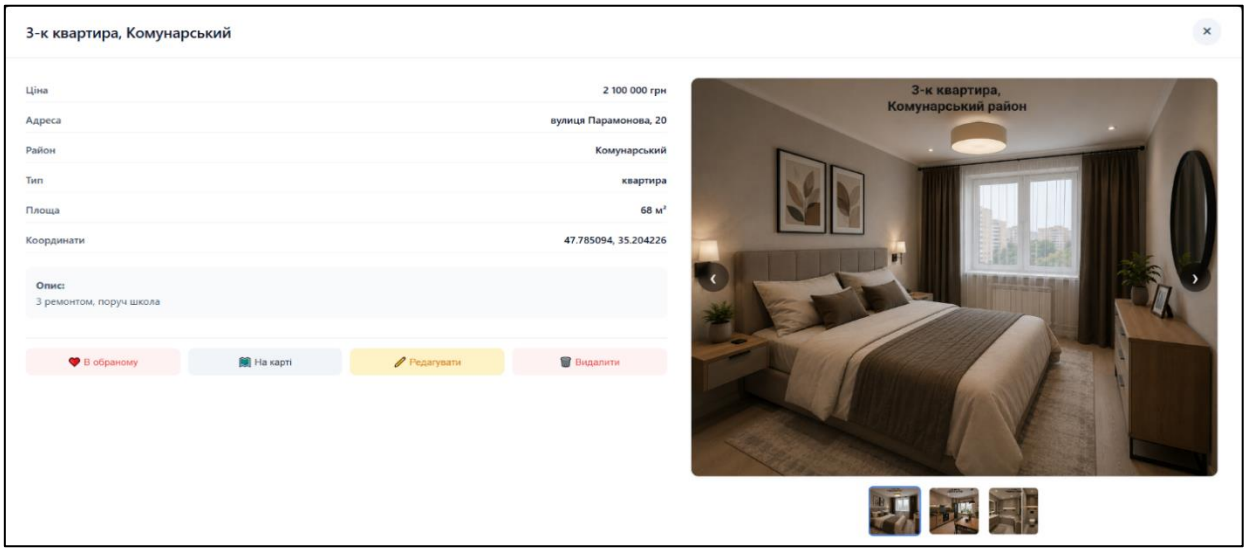


Рисунок 3.7 – Детальна панель об’єкта

На десктопних пристроях панель має двоколонкове компонування. Ліва колонка містить повну текстову інформацію: назву об’єкта, тип нерухомості, район, повну адресу, ціну, площу, координати (широту та довготу) та розгорнутий опис, якщо він був доданий адміністратором. Тут же розташована кнопка «В обране», «Видалити з обраного», стан якої змінюється динамічно залежно від того, чи доданий об’єкт до списку обраних поточним користувачем.

Права колонка відведена під галерею зображень. Якщо для об’єкта завантажено кілька фотографій, вони відображаються у вигляді великого зображення зі стрілками навігації ліворуч і праворуч. Користувач може гортати фотографії, клацаючи по стрілках, використовуючи колесико миші або проводячи пальцем на сенсорних пристроях. При кліку на зображення воно відкривається у повноекранному режимі (лайтбокс) на затемненому фоні, де також доступна навігація. Закриття повноекранного перегляду відбувається кліком на хрестик, на затемнену область або натисканням клавіші Escape. У верхній частині панелі є кнопка «Назад», яка повертає до попереднього режиму (карта або список). Стан відкритої панелі зберігається в URL-хеші, що дозволяє поділитися посиланням або відновити перегляд після оновлення сторінки.

Система підтримує реєстрацію нових користувачів та вхід до існуючого облікового запису. Форма реєстрації (див. рис. 3.8) містить поля «Ім’я користувача» та «Пароль». При спробі зареєструвати ім’я, яке вже існує в базі, сервер повертає помилку, що відображається у формі. Пароль проходить валідацію на мінімальну довжину. Форма входу аналогічна і містить посилання для переходу до реєстрації.

Після успішної автентифікації користувач отримує доступ до персональних функцій: кнопка «В обране» стає активною на всіх картках, маркерах та в детальній панелі. Кнопка входу в меню змінюється на ім’я користувача, а поруч з’являється кнопка виходу. Адміністратор додатково бачить кнопку «Додати об’єкт» та кнопки редагування й видалення на всіх об’єктах.

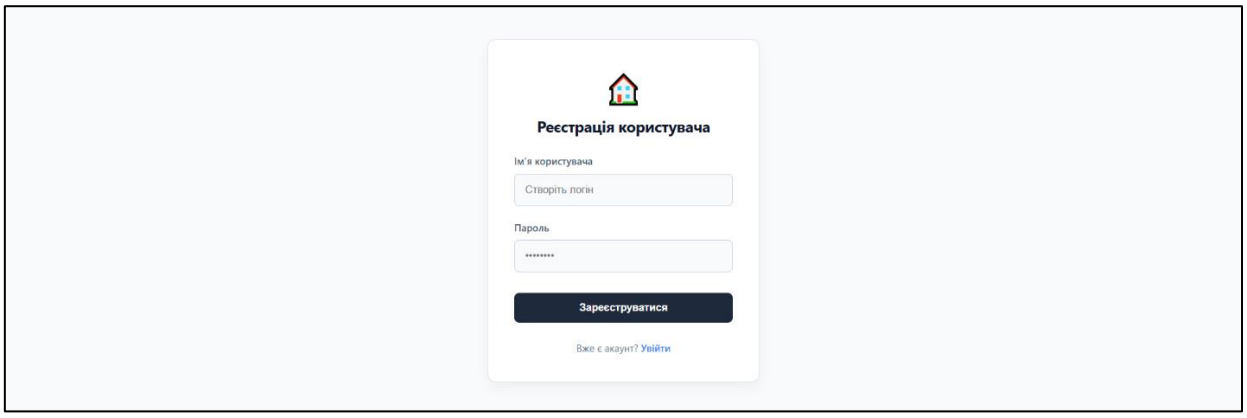


Рисунок 3.8 – Форма реєстрації

Адміністратор системи має доступ до повного циклу керування об’єктами нерухомості. Форма додавання нового об’єкта (рис. 3.9) відкривається при натисканні кнопки «Додати об’єкт» у верхньому меню. Вона містить поля: назва, опис, ціна, площа, випадаючий список типу нерухомості, випадаючий список району та адресу. Визначення координат реалізовано двома способами.

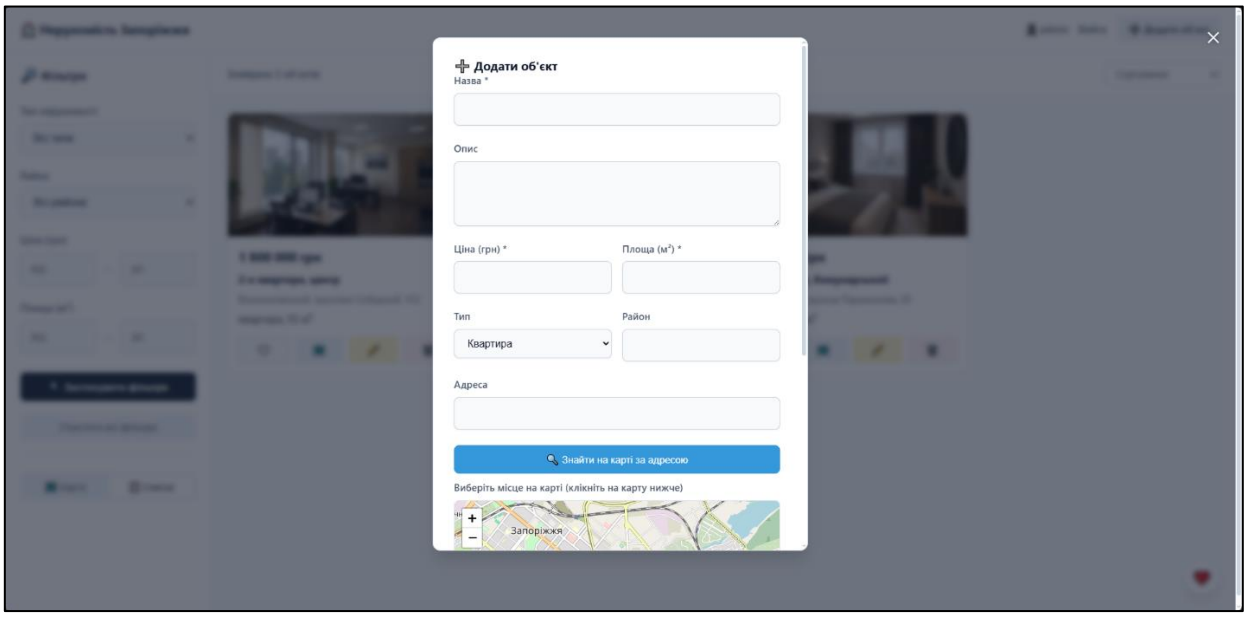


Рисунок 3.9 – Форма додавання об’єкта

По-перше, адміністратор може ввести адресу в текстове поле і натиснути кнопку геокодування – система надішле запит до сервісу PositionStack або Nominatim, отримає координати та автоматично заповнить поля широти й

довготи, а також встановить маркер на вбудованій міні-карті. По-друге, можна вказати координати вручну, просто клікнувши в потрібному місці на міні-карті; маркер переміститься, а поля координат оновляться. Такий дублюючий механізм компенсує можливі неточності безкоштовних геокодерів.

Для завантаження зображень передбачено поле вибору файлів із можливістю множинного вибору. Після вибору файлів під полем з'являються мініатюри завантажених зображень із кнопками видалення окремих фото. Максимальний розмір одного файлу – 5 МБ, дозволені типи – JPEG, PNG, GIF. Перевірка здійснюється не за розширенням, а за магічними байтами файлу, що гарантує безпеку.

Форма редагування об'єкта (рис. 3.10) викликається кнопкою «Редагувати» на картці або маркері. Вона містить ті самі поля, що й форма додавання, але попередньо заповнені актуальними даними. Додатково в ній відображається галерея вже завантажених зображень.

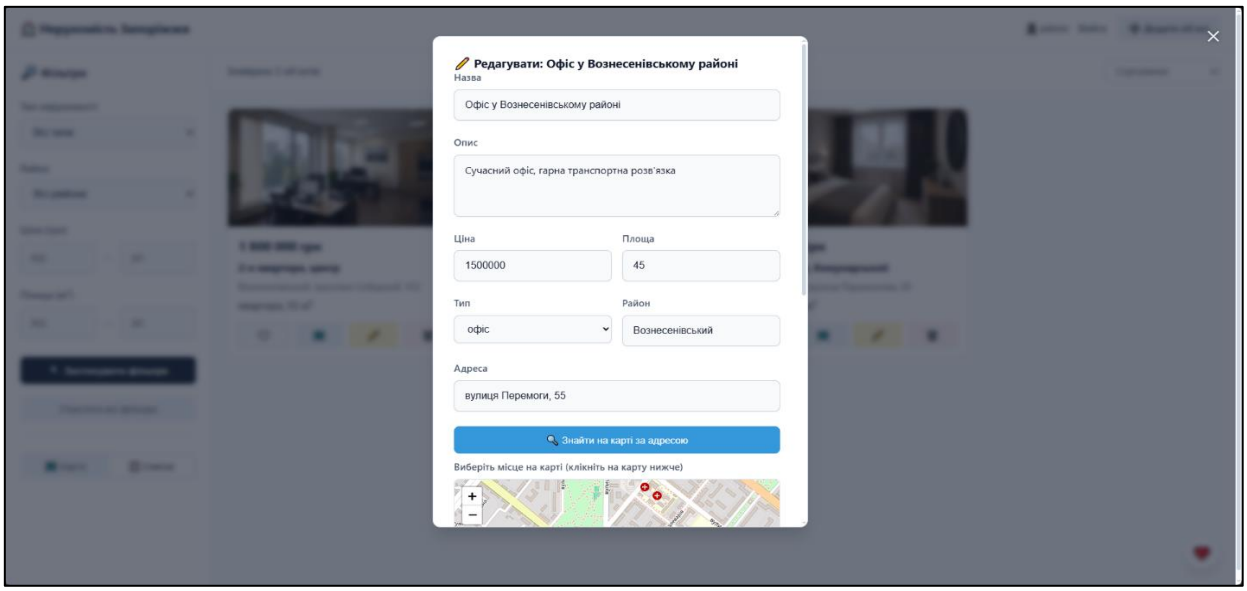


Рисунок 3.10 – Форма редагування об'єкта з галереєю

Користувач може видалити будь-яке з існуючих фото, додати нові, а також змінити порядок фотографій простим перетягуванням мишею завдяки інтеграції бібліотеки SortableJS [12]. Новий порядок автоматично зберігається при оновленні об'єкта.

Видалення об’єкта відбувається через кнопку «Видалити». Щоб уникнути випадкових дій, система запитує підтвердження через модальне діалогове вікно. При підтвердженні об’єкт разом з усіма пов’язаними зображеннями видаляється з бази даних, а маркер та картка миттєво зникають з інтерфейсу без перезавантаження сторінки.

Список обраних об’єктів доступний через плаваючу кнопку із сердечком в нижньому правому куті. При натисканні відкривається модальне вікно (рис. 3.11), яке містить перелік збережених користувачем об’єктів у вигляді компактних карток із назвою, ціною та мініатюрою. Кожна картка має кнопку видалення з обраного та є клікабельною – натискання відкриває детальну панель відповідного об’єкта. Якщо список порожній, виводиться відповідне повідомлення. Вікно закривається кліком на хрестик або на затемнену область навколо нього.

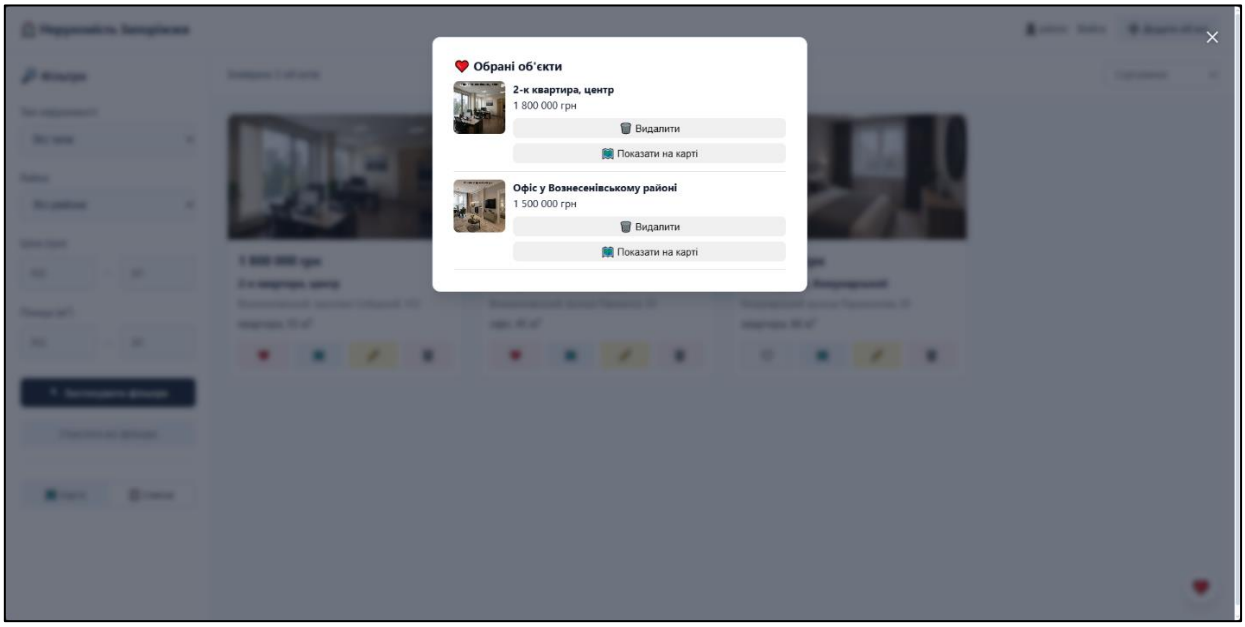


Рисунок 3.11 – Модальне вікно «Обране»

Інтерфейс системи повністю адаптований для роботи на пристроях із різною роздільною здатністю екрана, що досягається за допомогою медіа-запитів CSS. При ширині екрана менше 768 пікселів ліва панель фільтрів приховується за лівий край, а на її місці з’являється напівкругла кнопка-свайп. Натискання на цю кнопку або жест змахування спричиняє плавне виїжджання

панелі поверх карти з напівпрозорим темним оверлеєм. Закрити панель можна кнопкою з хрестиком у її верхньому куті або кліком на затемнену область. Мобільна версія з виїзною панеллю фільтрів показана на рис. 3.12.

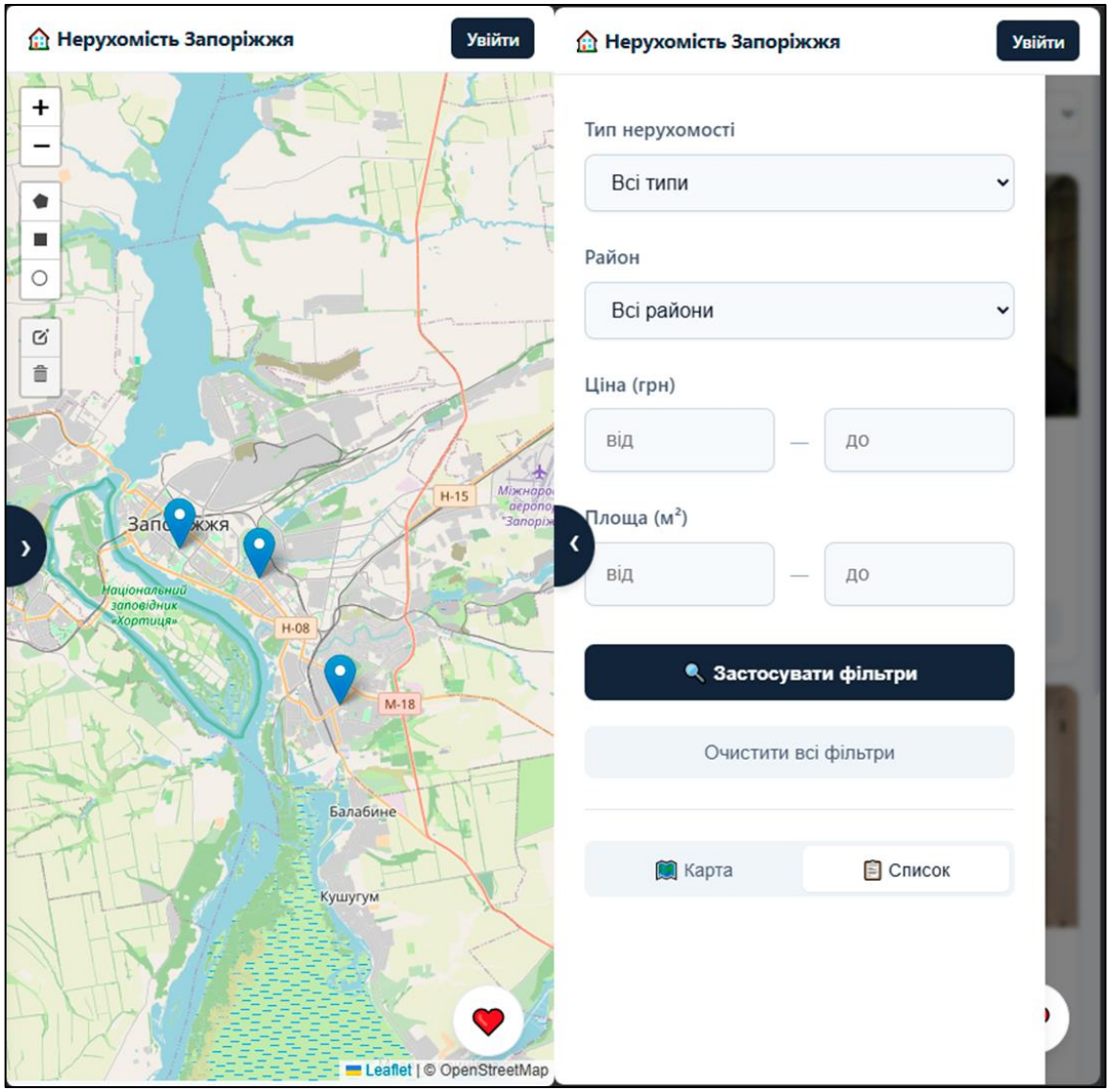


Рисунок 3.12 – Мобільна версія

Детальна панель об’єкта на мобільних пристроях змінює компонування: зображення розташовуються зверху, а вся текстова інформація – під ними, що забезпечує зручне гортання вмісту однією рукою. Кнопки навігації в галереї мають збільшений розмір, комфортний для натискання пальцем. Модальні вікна займають майже всю площу екрана, не виходячи за його межі, і мають внутрішню прокрутку, якщо вміст перевищує висоту екрана. Приклад відкритої детальної панелі на смартфоні – на рис. 3.13.

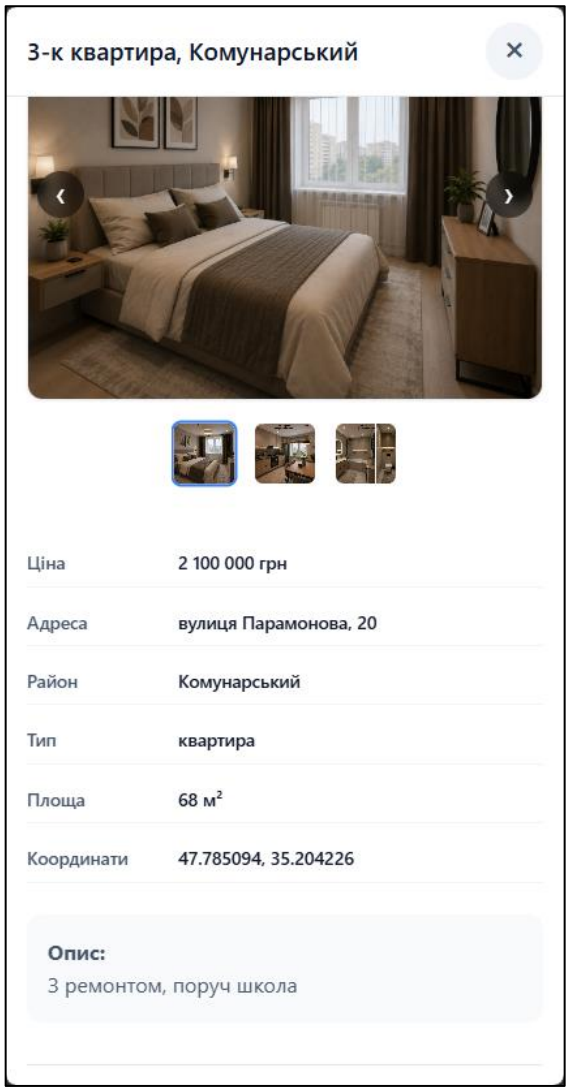


Рисунок 3.13 – Мобільна версія

3.4 Тестування системи

Тестування розробленої геоінформаційної системи проводилося за кількома напрямками: функціональне тестування відповідності вимогам, тестування просторових запитів, перевірка адаптивності інтерфейсу та тестування безпеки.

У таблиці 3.1 наведено основні функціональні тест-кейси з фіксацією очікуваного та фактичного результатів, а також підсумкового статусу.

Функціональне тестування підтвердило, що всі заявлені можливості системи працюють коректно відповідно до специфікації вимог.

Таблиця 3.1 – Результати функціонального тестування

№	Тест-кейс	Очікуваний результат	Фактичний результат
1	Завантаження головної сторінки	Карта відображається з центром у Запоріжжі, панель фільтрів видима	Виконано
2	Застосування фільтра за типом «квартира»	На карті залишаються лише квартири, у списку теж	Виконано
3	Фільтр за ціною від 500000 до 2000000 грн	Відображаються об’єкти в заданому діапазоні	Виконано
4	Малювання полігону на карті	Після завершення малювання показуються об’єкти всередині полігону	Виконано
5	Клік по маркеру	Відкривається детальна панель з інформацією про об’єкт	Виконано
6	Додавання об’єкта в «обране»	Об’єкт з’являється у модальному вікні «Обране»	Виконано
7	Адміністратор додає новий об’єкт через форму	Об’єкт з’являється на карті та у списку	Виконано
8	Адміністратор додає фото до об’єкта	Фото відображається в галереї детальної панелі	Виконано
9	Адміністратор видаляє фото	Фото зникає з галереї	Виконано
10	Перемикання між картою та списком	Вигляд змінюється без помилок	Виконано

Тестування просторових запитів включало перевірку коректності роботи алгоритму полігонального пошуку для різних форм полігонів: опуклих, увігнутих, таких, що містять значну кількість об’єктів, і таких, що не містять жодного. У всіх випадках система правильно ідентифікувала входження точки в полігон. Час виконання запиту для бази з 500 об’єктів не перевищував 0,4 секунди, що цілком відповідає встановленій вимозі.

Додатково перевірено роботу геокодерів PositionStack та Nominatim: при введенні коректної адреси координати визначаються з прийнятною точністю, при помилковій адресі система повертає повідомлення про неможливість геокодування та пропонує вказати координати вручну [9].

Тестування адаптивності інтерфейсу проводилося на емуляторах мобільних пристроїв із різною роздільною здатністю екрана. На всіх пристроях із шириною менше 768 пікселів панель фільтрів коректно ховалася та виїжджала за натисканням кнопки-свайпу, детальна панель об’єкта перебудовувалася у вертикальне компонування, а всі інтерактивні елементи зберігали достатній розмір для комфортного натискання пальцем.

Перевірка безпеки охоплювала спроби доступу неавторизованого користувача до адміністративних маршрутів – усі такі спроби були відхилені з HTTP-кодом 403. Спроба завантажити файл із розширенням .jpg, але з вмістом текстового файлу, була успішно заблокована завдяки перевірці магічних байтів. Паролі користувачів у базі даних зберігаються виключно у хешованому вигляді. Ручна перевірка на стійкість до SQL-ін’єкцій шляхом підстановки спеціальних символів у поля фільтрів підтвердила, що всі запити коректно параметризуються, і шкідливий код не виконується.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра було реалізовано повний цикл створення геоінформаційної системи для візуалізації та аналізу об’єктів нерухомості. Досягнуто поставленої мети – підвищення швидкості та зручності пошуку об’єктів нерухомості для кінцевого користувача шляхом створення веборієнтованої геоінформаційної системи з просторовою фільтрацією та інтерактивною картою.

У ході роботи послідовно вирішено всі завдання, сформульовані у вступі. Виконано аналіз предметної області та огляд існуючих геоінформаційних систем, виявлено їхні недоліки та сформульовано функціональні й нефункціональні вимоги. Обґрунтовано вибір трирівневої клієнт-серверної архітектури з використанням Flask, Leaflet.js, SQLite та Shapely.

Розроблено структуру бази даних із п’яти нормалізованих таблиць, створено індекси для пришвидшення фільтрації та реалізовано зв’язки із зовнішніми ключами. Клієнтську частину реалізовано як односторінковий вебзастосунок з адаптивним дизайном, панеллю фільтрів, двома режимами перегляду та детальною панеллю об’єкта. Забезпечено просторовий аналіз через полігональний пошук на сервері за допомогою Shapely, а атрибутивну фільтрацію – через параметризовані SQL-запити. Впроваджено автентифікацію користувачів із розмежуванням ролей, механізм обраного з оптимістичним оновленням інтерфейсу та адміністративний інтерфейс для повного циклу керування об’єктами й зображеннями. Проведено функціональне тестування, тестування продуктивності, безпеки та сумісності, які підтвердили відповідність системи всім вимогам. Зокрема, полігональний пошук для тисячі об’єктів виконується менш ніж за 0,4 секунди, а система є стійкою до SQL-ін’єкцій та XSS-атак.

ПЕРЕЛІК ПОСИЛАНЬ

1. Flask Documentation. URL: https://flask.palletsprojects.com/ (дата звернення: 20.03.2026).

2. Leaflet.js – JavaScript бібліотека для інтерактивних карт. URL: https://leafletjs.com (дата звернення: 22.03.2026).

3. SQLite Documentation. URL: https://sqlite.org/docs.html (дата звернення: 25.03.2026).

4. Shapely Documentation. URL: https://shapely.readthedocs.io/ (дата звернення: 28.03.2026).

5. Flask-Login Documentation. URL: https://flask-login.readthedocs.io/ (дата звернення: 02.04.2026).

6. Flask-Bcrypt Documentation. URL: https://flask-bcrypt.readthedocs.io/ (дата звернення: 05.04.2026).

7. OpenStreetMap. URL: https://www.openstreetmap.org/ (дата звернення: 08.04.2026).

8. PositionStack Geocoding API. URL: https://positionstack.com/ (дата звернення: 12.04.2026).

9. Nominatim Geocoding. URL: https://nominatim.org/ (дата звернення: 15.04.2026).

10. Документація Leaflet.markercluster. URL: https://github.com/Leaflet/Leaflet.markercluster (дата звернення: 18.04.2026).

11. Документація Leaflet.draw. URL: https://github.com/Leaflet/Leaflet.draw (дата звернення: 20.04.2026).

12. SortableJS – бібліотека для перетягування елементів. URL https://sortablejs.github.io/Sortable/ (дата звернення: 23.04.2026).

ДОДАТОК А

SQL-код створення таблиць бази даних

```
CREATE TABLE IF NOT EXISTS property (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    description TEXT,
    price REAL,
    area REAL,
    property_type TEXT,
    district TEXT,
    address TEXT,
    lat REAL NOT NULL,
    lng REAL NOT NULL
);
CREATE TABLE IF NOT EXISTS property_images (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    property_id INTEGER NOT NULL,
    image_url TEXT NOT NULL,
    sort_order INTEGER DEFAULT 0,
    FOREIGN KEY (property_id) REFERENCES property(id) ON DELETE CASCADE
);
CREATE TABLE IF NOT EXISTS user (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    role TEXT DEFAULT 'user'
);
CREATE TABLE IF NOT EXISTS favorite (
    user_id INTEGER,
    property_id INTEGER,
    PRIMARY KEY (user_id, property_id),
    FOREIGN KEY (user_id) REFERENCES user(id),
    FOREIGN KEY (property_id) REFERENCES property(id)
);
CREATE TABLE IF NOT EXISTS search_log (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    query TEXT,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    ip_address TEXT
);
```

ДОДАТОК Б

Основні маршрути Flask-сервера

```
import os, sqlite3
from flask import Flask, request, jsonify, render_template, redirect, url_for
from flask_login import login_user, login_required, logout_user, current_user
from flask_bcrypt import Bcrypt
from werkzeug.utils import secure_filename
from shapely.geometry import Point, Polygon
from models import User

app = Flask(__name__)
bcrypt = Bcrypt(app)

#Підключення до бази даних
def get_db():
    conn = sqlite3.connect('properties.db')
    conn.row_factory = sqlite3.Row
    return conn

#Декоратор для адміністратора
def admin_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        if not current_user.is_authenticated or current_user.role != 'admin':
            return jsonify({'error': 'Доступ заборонено'}), 403
        return f(*args, **kwargs)
    return decorated

# --- Головна сторінка ---
@app.route('/')
def index():
    return render_template('index.html')

#Отримання списку об'єктів з фільтрацією та сортуванням
@app.route('/api/properties')
def get_properties():
    conn = get_db()
    c = conn.cursor()
    query = "SELECT * FROM property WHERE 1=1"
    params = []
```

```
    if request.args.get('type'):
        query += " AND property_type = ?"
        params.append(request.args['type'])
    if request.args.get('district'):
        query += " AND district = ?"
        params.append(request.args['district'])
    if request.args.get('price_min'):
        query += " AND price >= ?"
        params.append(float(request.args['price_min']))
    if request.args.get('price_max'):
        query += " AND price <= ?"
        params.append(float(request.args['price_max']))
    if request.args.get('area_min'):
        query += " AND area >= ?"
        params.append(float(request.args['area_min']))
    if request.args.get('area_max'):
        query += " AND area <= ?"
        params.append(float(request.args['area_max']))

    c.execute(query, params)
    rows = c.fetchall()
    # Сортування на стороні сервера
    sort = request.args.get('sort', '')
    if sort == 'price_asc':
        rows = sorted(rows, key=lambda r: safe_price_value(r['price']))
    elif sort == 'price_desc':
        rows = sorted(rows, key=lambda r: safe_price_value(r['price']), reverse=True)
    # Додавання зображень та статусу обраного
    result = []
    if current_user.is_authenticated:
        c.execute("SELECT property_id FROM favorite WHERE user_id=?", (current_user.id,))
        fav_ids = {row['property_id'] for row in c.fetchall()}
    else:
        fav_ids = set()
    for r in rows:
        d = dict(r)
        c.execute("SELECT image_url FROM property_images WHERE property_id=? ORDER
BY sort_order", (d['id'],))
        d['images'] = [row['image_url'] for row in c.fetchall()]
        d['is_favorite'] = d['id'] in fav_ids
        result.append(d)
    conn.close()
    return jsonify(result)
```

```
#Отримання списку районів
@app.route('/api/districts')
def get_districts():
    conn = get_db()
    c = conn.cursor()
    c.execute("SELECT DISTINCT district FROM property WHERE district IS NOT NULL
AND district != """)
    rows = [row['district'] for row in c.fetchall()]
    conn.close()
    return jsonify(rows)

#Додавання нового об'єкта
@app.route('/api/properties', methods=['POST'])
@login_required
@admin_required
def add_property():
    title = request.form.get('title')
    description = request.form.get('description', '')
    price = request.form.get('price', type=float) or 0
    area = request.form.get('area', type=float)
    property_type = request.form.get('property_type')
    district = request.form.get('district', '')
    address = request.form.get('address', '')
    lat = request.form.get('lat', type=float)
    lng = request.form.get('lng', type=float)
    conn = get_db()
    c = conn.cursor()
    c.execute('''INSERT INTO property (title, description, price, area, property_type, district,
address, lat, lng)
                 VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)''',
              (title, description, price, area, property_type, district, address, lat, lng))
    conn.commit()
    new_id = c.lastrowid
    conn.close()
    return jsonify({'id': new_id}), 201

#Додавання фотографії до об'єкта
@app.route('/api/properties/<int:prop_id>/images', methods=['POST'])
@login_required
@admin_required
def add_property_image(prop_id):
    file = request.files.get('image')
    if not file or file.filename == '':
        return jsonify({'error': 'Файл не обрано'}), 400
```

```
    if not allowed_image(file):
        return jsonify({'error': 'Недопустимий формат файлу'}), 400
    filename = secure_filename(file.filename)
    name, ext = os.path.splitext(filename)
    unique_filename = f"{name}_{os.urandom(4).hex()}{ext}"
    file.save(os.path.join(app.config['UPLOAD_FOLDER'], unique_filename))
    image_url = '/static/uploads/' + unique_filename
    conn = get_db()
    c = conn.cursor()
    c.execute("SELECT COALESCE(MAX(sort_order), -1)+1 FROM property_images WHERE
property_id=?", (prop_id,))
    next_order = c.fetchone()[0]
    c.execute("INSERT INTO property_images (property_id, image_url, sort_order) VALUES (?,
?, ?)",
              (prop_id, image_url, next_order))
    conn.commit()
    new_id = c.lastrowid
    conn.close()
    return jsonify({'id': new_id, 'image_url': image_url}), 201

#Редагування об'єкта
@app.route('/api/properties/<int:prop_id>', methods=['PUT'])
@login_required
@admin_required
def update_property(prop_id):
    data = request.form if request.form else request.get_json()
    if not data:
        return jsonify({'error': 'Некоректний формат даних'}), 400
    title = data.get('title')
    description = data.get('description', '')
    price = data.get('price') or 0
    area = data.get('area')
    property_type = data.get('property_type')
    district = data.get('district', '')
    address = data.get('address', '')
    lat = data.get('lat')
    lng = data.get('lng')
    if not title or area is None or lat is None or lng is None:
        return jsonify({'error': "Обов'язкові поля не заповнені"}), 400
    conn = get_db()
    c = conn.cursor()
    c.execute('''UPDATE property SET title=?, description=?, price=?, area=?, property_type=?,
                 district=?, address=?, lat=?, lng=? WHERE id=?''',
              (title, description, price, area, property_type, district, address, lat, lng, prop_id))
```

```
    conn.commit()
    conn.close()
    return jsonify({'id': prop_id, 'updated': True})

#Видалення об'єкта
@app.route('/api/properties/<int:prop_id>', methods=['DELETE'])
@login_required
@admin_required
def delete_property(prop_id):
    conn = get_db()
    c = conn.cursor()
    c.execute("SELECT image_url FROM property_images WHERE property_id=?", (prop_id,))
    for img in c.fetchall():
        delete_old_image(img['image_url'])     # видалення файлів
    c.execute("DELETE FROM property_images WHERE property_id=?", (prop_id,))
    c.execute("DELETE FROM property WHERE id=?", (prop_id,))
    conn.commit()
    conn.close()
    return jsonify({'id': prop_id, 'deleted': True})

#Полігональний пошук
@app.route('/api/properties_by_polygon', methods=['POST'])
def properties_by_polygon():
    data = request.get_json()
    if not data or 'polygon' not in data:
        return jsonify([])
    coords = data['polygon']
    try:
        poly = Polygon(coords)
    except:
        return jsonify({'error': 'Некоректний полігон'}), 400
    conn = get_db()
    c = conn.cursor()
    c.execute("SELECT * FROM property")
    rows = c.fetchall()
    result = []
    for r in rows:
        d = dict(r)
        c2 = conn.cursor()
        c2.execute("SELECT image_url FROM property_images WHERE property_id=? ORDER
BY sort_order", (d['id'],))
        d['images'] = [row['image_url'] for row in c2.fetchall()]
        c2.close()
        if poly.contains(Point(r['lng'], r['lat'])):
```

```
            result.append(d)
    conn.close()
    return jsonify(result)

# Обране
@app.route('/api/favorites', methods=['GET'])
@login_required
def get_favorites():
    conn = get_db()
    c = conn.cursor()
    c.execute('''SELECT p.*, 1 as is_favorite FROM property p
                 JOIN favorite f ON p.id = f.property_id WHERE f.user_id = ?''', (current_user.id,))
    rows = c.fetchall()
    result = []
    for r in rows:
        d = dict(r)
        c.execute("SELECT image_url FROM property_images WHERE property_id=? ORDER
BY sort_order", (d['id'],))
        d['images'] = [row['image_url'] for row in c.fetchall()]
        result.append(d)
    conn.close()
    return jsonify(result)

@app.route('/api/favorites', methods=['POST'])
@login_required
def toggle_favorite():
    prop_id = request.json.get('property_id')
    conn = get_db()
    c = conn.cursor()
    try:
        c.execute("SELECT * FROM favorite WHERE user_id=? AND property_id=?",
(current_user.id, prop_id))
        if c.fetchone():
            c.execute("DELETE FROM favorite WHERE user_id=? AND property_id=?",
(current_user.id, prop_id))
            action = 'removed'
        else:
            c.execute("INSERT INTO favorite (user_id, property_id) VALUES (?, ?)",
(current_user.id, prop_id))
            action = 'added'
    except sqlite3.IntegrityError:
        c.execute("DELETE FROM favorite WHERE user_id=? AND property_id=?",
(current_user.id, prop_id))
        action = 'removed'
```

```
    conn.commit()
    conn.close()
    return jsonify({'property_id': prop_id, 'action': action})

#Реєстрація та автентифікація
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        hashed = bcrypt.generate_password_hash(password).decode('utf-8')
        conn = get_db()
        c = conn.cursor()
        try:
            c.execute("INSERT INTO user (username, password_hash) VALUES (?, ?)", (username,
hashed))
            conn.commit()
        except sqlite3.IntegrityError:
            flash("Користувач із таким ім'ям уже існує", 'danger')
            return render_template('register.html')
        conn.close()
        flash('Реєстрація успішна!', 'success')
        return redirect(url_for('login'))
    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        conn = get_db()
        c = conn.cursor()
        c.execute("SELECT * FROM user WHERE username=?", (username,))
        row = c.fetchone()
        conn.close()
        if not row or not bcrypt.check_password_hash(row['password_hash'], password):
            flash('Невірне ім’я або пароль', 'danger')
        else:
            user = User(row['id'], row['username'], row['role'])
            login_user(user)
            flash('Успішний вхід!', 'success')
            return redirect(url_for('index'))
    return render_template('login.html')
```

```
@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('index'))
```